بسم الله الرحمن الرحيم
قال تعالى " وقل رب زدني علماً "
(الجزء الأول)

التخصيص الساكن للذاكرة Static Memory Allocation:

عندما نصرح متحول ضمن برنامج مكتوب بلغة تربو باسكال فإن المترجم يعرف تماما كمية الذاكرة التي يحتاجها هذا المتحول حيث يُخصص allocate المترجم حجرات الذاكرة الخاصة بالمتحولات العامة global variable و الثوابت constant ضمن مقطع المعطيات data segment .

و مقطع المعطيات عبارة عن مساحة من الذاكرة محدودة الطول يحدد ويقرر حجمها بناءً على عدد و نوع المتحولات العامة و الثوابت المصرح عنها ضمن البرنامج أما المتحولات الموضعية (المحلية) local variables و الوسطاء parameters فإن المترجم يخصص لها حجرات في الذاكرة عندما يتطلب تنفيذ البرنامج ذلك.

لكن المترجم يحجز بشكل أولي كمية من الذاكرة تدعى بمقطع المكدس stack segment من أجل استخدامها لهذه المتحولات الموضعية (المحلية) و طول مقطع المكدس ثابت و يمكن الاستعانة بتوجيه المترجم {$M} لتحديد طول المكدس stack أثناء ترجمة البرنامج وإلا فإن الطول الافتراضي لهذا المكدس سوف يستخدم تلقائياً و استخدام مقطع المكدس هذا يعتمد على الروتينات التي يتوقف تنفيذها مؤقتاً لتنفيذ الروتينات التي تستدعي بعضها و هكذا.....دواليك.

يعتبر مقطع المعطيات و مقطع المكدس أمثلة عن التخصيص الساكن للذاكرة static memory allocation وقد استخدمنا صفة ساكن لأن كمية الذاكرة المخصصة يتم تحديدها أثناء ترجمة البرنامج و قبل تنفيذه و لأن هذه الكمية ثابتة لا تتغير أبداً.

مساوئ التخصيص الساكن للذاكرة drawbacks static memory Allocation:

قد لا نستطيع في بعض الأحيان تخصيص حجرات من الذاكرة بشكل ثابت. فمثلاً : افترض أننا نريد كتابة برامج لتصنيف معطيات عددية فإن هذا البرنامج سوف يحصل على المعطيات من المستخدم و لكن عدد هذه المعطيات الواجب تخزينها و تصنيفها ليس ثابت فمن اجل القيام بهذا العمل توجد طريقتان:

١. التصريح عن نسق ذي حجرات كثيرة بحيث تكفي لتخزين القيم المتوقع إدخالها من المستخدم و هذه الطريقة تستخدم التخصيص الساكن للذاكرة.
٢. استخدام التخصيص الديناميكي للذاكرة و ذلك باستخدام المؤشرات و التي هي موضوعنا في هذا الكتاب.

التخصيص الديناميكي للذاكرة Dynamic Memory Allocation:

إن عملية حجز حجر الذاكرة في التخصيص الديناميكي للذاكرة لا تتم أثناء الترجمة البرنامج و قبل تنفيذه و لكنها تحدث عندما يحتاج البرنامج أثناء عمله إلى حجرات من الذاكرة فإنه يخصص بعضها و يستخدمها ضمن البرنامج و حجرات الذاكرة هذه تقتطع أو تؤخذ من مساحة من الذاكرة تدعى بالكومة heap و هي مساحة من آخر الذاكرة يحدد موقعها تماماً بعد تحميل مقطع شفرة التعليمات code segment و مقطع المعطيات data segment و مقطع المكدس stack segment الخاصة بالبرنامج.

توضع طبقات البرنامج program layout:

يظهر لنا الشكل التالي رسماً تخطيطياً تقريبياً لتوضع طبقات البرنامج أثناء تنفيذه حيث يحتوي على المكونات الأساسية لحيز الذاكرة الذي يشغله البرنامج أثناء التنفيذ. و هذا التو ضع لمحتويات الذاكرة قد يختلف من برنامج إلى آخر بسبب بعض الصفات الخاصة مثل :التراكب overlays و حجم و عدد المتحولات و الثوابت المستخدمة ضمن البرنامج.

تعد الذاكرة العنصر الأهم أثناء تنفيذ البرنامج لذلك يرتب نظام التشغيل DOS و يخزن المعلومات فيها ووفق الترتيب المذكور في الشكل التالي فإذا كنا نعمل تحت النظام DOS فإن جزءاً من نظام التشغيل سوف يخزن في الذاكرة وسوف تخزن البرامج الأخرى المحملة في الذاكرة أثناء تنفيذ البرنامج أيضاً في القسم نفسه و الذي يشغل الجزء السفلي من ذاكرة نظام التشغيل DOS و كمثال عن هذه البرامج برنامج بيئة التطوير المتكاملة IDE و هو عبارة عن محرر نصوص إضافة إلى مترجم تربو باسكال .فإذا كنا نعمل ضمن هذا البرنامج فإن هذا البرنامج سوف يخزن أيضاً في نفس القسم من الذاكرة الذي ذكرته سابقاً و هذا القسم يدعى بقسم البرامج المتنوعة miscellaneous.

أعلى ذاكرة DOS

(مساحة غير مستخدمة من الذاكرة) ↑ ما تبقى من الذاكرة يدعى بالكومة و التي تستخدم للتخصيص الديناميكي للمتحولات
يتوضع هنا مقطع المكدس stack segment و الذي يمثل المتحولات الموضعية و الوسطاء ↓ (مساحة غير مستخدمة من المكدس)
يتوضع هنا مقطع شفرة المعطيات data segment و الذي يمثل المتحولات العامة و الثوابت
يتوضع هنا مقطع شفرة التعليمات code segment و الذي يمثل تعليمات البرنامج
يستخدم النظام DOS هذا الجزء لأغراضه الخاصة و تستخدمه أيضاً بعض البرامج miscellaneous

أسفل ذاكرة DOS

مقاطع البرنامج program segments:

بعد أن وضِعت البرامج السابقة في القسم السفلي من الذاكرة يوضع مقطع شفرة التعليمات الخاص ببرنامجنا ومن ثم يُتبع هذا المقطع بمقطع المعطيات و مقطع المكدس و هذه المقاطع ثابتة الطول و يتعلق طولها بالبرنامج نفسه حي تحدد كمية تعليمات البرنامج حجم مقطع شفرة التعليمات و يحدد حجم المتحولات العامة و الثوابت حجم مقطع المعطيات و هكذا يتعلق حجم المكدس المستخدم في البرنامج بالوسطاء الموجودة في البرنامج و البيئات المحلية للروتينات التي يتوقف عملها مؤقتاً من أجل نقل التحكم إلى الروتينات الأخرى.

و مقطع المكدس هذا له خصوصية في عمله إذ تخزن فيه المعلومات آنياً حتى تُطلب من جديد ولذلك يتغير الحجم المستخدم من هذا المقطع حسب عدد الروتينات المستدعاة فيزداد حجمه و ينقص تبعاً لذلك حيث يزداد grows طول المكدس عكسياً أي من العنوان الأعلى إلى العنوان الأدنى.وقد مثلنا هذه الخاصية للمكدس بالسهم المتجه إلى الأسفل ضمن الشكل السابق.

عندما ينتهي تنفيذ روتين ما يفرغ جزء المكدس الذي استخدم لتخزين البيئة المحلية لهذا الروتين و تعاد هذه المعلومات لاستخدمها من قبل البرنامج . و في حال احتاج البرنامج حجماً أكبر من الحجم المتاح له (المحدد أثناء ترجمة البرنامج) فإن البرنامج سوف يتوقف تنفيذه مع خطأ هو حدوث طفحان (فيض) في المكدس stack overflow و حدوث مثل هذا الخطأ يعتمد على حالة فحص المكدس هل هي محفزة أم لا و التي يتحكم بها توجيه المترجم {$S} فإذا كانت حالة فحص المكدس فعالة فإن البرنامج يفحص وجود مساحة في المكدس تكفي لتخزين مستلزمات الروتين قبل البدء بتنفيذه و إلا فإن البرنامج سوف ينتهي مع خطأ . أما إذا كانت حالة فحص المكدس هي عدم الفحص فإن البرنامج لن يفحص المكدس قبل تحميل البيئة الموضعية للروتين و في هذه الحالة لن يفشل تنفيذ البرنامج وحده بل سيفشل معه النظام الداخلي للحاسب و سوف يستدعي هذا إعادة إقلاع reboot الحاسب من جديد أو إيقاف البرنامج.

وبشكل نظامي فإن حالة فحص المكدس تكون محفزة فإذا أردنا تغيير حالة الفحص تلك يمكننا إضافة أحد التوجيهين التاليين:

{$S+} (*يحفز حالة فحص المكدس*)

{$S-} (*يلغي حالة فحص المكدس *)

كومة البرنامج the program heap:

الكومة بالتعريف هي المساحة من الذاكرة المتبقية دون استخدام و التي تقع بعد مقاطع البرنامج بعد تحميلها فإذا كان البرنامج المحمل صغيراً و يملك القليل من المعطيات العامة و لا توجد برامج ضخمة محملة في الذاكرة فإن الكومة تلك ستكون كبيرة جداً و العكس صحيح .
يبين لنا البرنامج التالي طريقة استخدام تابعين مسبقي التعريف مهمتها تزويدنا بمعلومات حول ذاكرة الكومة:

```
code
program test;
begin
write('memavail = ',memavail:7,' bytes;  ');
write('maxavail = ',maxavail:7,' bytes; ');
readln;
end.
```

يعيد التابع memavail قيمة صحيحة طويلة longint تمثل عدد البايتات من الذاكرة التي تستطيع الكومة استخدامها و هذه البايتات ليست متعاقبة بالضرورة.
أما لتابع maxavail فإنه يعيد أيضاً قيمة صحيحة طويلة ولكنها تمثل حجم أكبر حيز متعاقب تستطيع الكومة استخدامه و هذا الحجم قد يتغير حسب كمية الذاكرة المتبقية و الخاصة بالكومة و التي تتغير إذا كان البرنامج يستخدم التخصيص الديناميكي للذاكرة.
تدار عناصر الكومة في لغة تربو باسكال بواسطة مكتبة تدعى بمدير الكومة heap manager و مدير الكومة هذا يحتفظ دائماً بقائمة تتضمن حالة مساحات التخزين المخصصة للكومة (طبعا؟؟ ☹ فهو المدير).

المؤشرات pointers:

المؤشرات عبارة عن نوع معطيات يشير إلى point to موضع معين في الذاكرة و عليه فإن المتحول من النوع مؤشر يحتوي على العنوان الذي خزنت فيه قيمة هذا المتحول. و هذا العنوان يتعلق بمواضع التخزين في الذاكرة و المسموح استخدامها ضمن البرنامج و التي تستطيع المؤشرات الوصول إليها. و حجم مواضع التخزين يدعى تخصيصاً ديناميكياً لأن التخصيص يحدث عندما يحتاج البرنامج إلى ذلك .
البرنامج التالي يوضح لنا كيفية التعريف و التصريح عن المؤشرات و كيف يمكننا الحصول على المعلومات التي تشير إليها هذه المؤشرات:

```
code
program test;
type
intptr=^integer;
realptr=^real;
var ip:intptr;
rp:realptr;
isize,rsize:integer;
begin
write('memavail = ',memavail:7,' bytes;  ');
writeln('maxavail = ',maxavail:7,' bytes; ');
new(rp);
rsize:=sizeof(rp^);
writeln('after allocating ',rsize:7,'bytes: ');
write('memavail = ',memavail:7,' bytes;  ');
writeln('maxavail = ',maxavail:7,' bytes; ');
```

إعداد المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com

```
new(ip);
isize:=sizeof(ip^);
writeln('after allocating ',isize:7,' bytes: ');
write('memavail = ',memavail:7,' bytes;  ');
writeln('maxavail = ',maxavail:7,' bytes; ');
readln
end.
readln;
end.
```

خرج
البرنامج شبيه بالخرج التالي:

```
memavail =  539728 bytes;  maxavail =  539728 bytes;
after allocating      6bytes:
memavail =  539720 bytes;  maxavail =  539720 bytes;
after allocating       2 bytes:
memavail =  539712 bytes;  maxavail =  539712 bytes;
```

يُطلق على المساحة من الذاكرة و التي تحتوي على قيمة المؤشر target أما محتويات المؤشر فتمثل عنوان غاية المؤشر و سوف أشرح هذه المفاهيم.

تعريف نوع المعطيات مؤشر defining pointer type:

يحتوي البرنامج السابق على نوعي معطيات مؤشرهما:

intptr=^integer;
realptr=^real;

فُقد عرف النوع intptr على أنه مؤشر من النوع الصحيح أي العنوان أن العنوان المخزن في المتحول من النوع intptr يُفهم على أنه عنوان موضع في الذاكرة يتألف من حجرتين و محتويات هاتين الحجرتين سوف تُفسر على أنها أعداد صحيحة . وبشكل مشابه عرف النوع realptr على انه مؤشر من النوع الحقيقي وهذا يعني أن العنوان المخزن في المتحول من النوع realptr سوف يُفهم على أنه عنوان بداية مساحة من الذاكرة بطول ستة بايتات و سوف تحتوي هذه البايتات الست على عدد حقيقي.

و الصيغة الكتابية لتعريف المؤشرات هي:

<نوع المعطيات الأساسي الخاص بغاية المؤشر>"^"=<مميز>

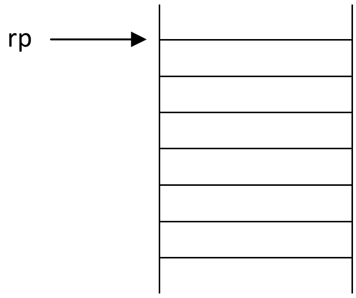
يشير الرمز ^ الذي يسبق نوع المعطيات الأساسي إلى أننا نُعرف مؤشراً ونوع المعطيات الأساسي لهذا المؤشر يمكن أن يكون بسيطا أو مركباً و في الحقيقة يمكن أن يكون أكثر تعقيدا أي يمكن أن يكون مؤشراً أيضاً و نوع المعطيات هذا يجب تحديده أثناء تعريف نوع المعطيات مؤشر حتى يعرف البرنامج مسبقاً كمية الذاكرة الواجب تخصيصها لتخزين غاية المؤشر.

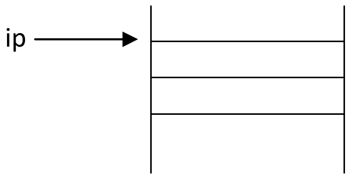
التصريح عن متحولات من نوع مؤشر declaring pointer variables:

بعد أن نحدد نوع المعطيات الأساسي للمؤشر يمكننا التصريح عن متحولات تنتمي لنوع المعطيات مؤشر هذا و الصيغة الكتابية لهذا التصريح تطابق مثيلاتها بالنسبة لأنواع المعطيات الأخرى.

<نوع المعطيات مؤشر>:<مميز>

لقد صرحنا عن المتحولين ip و rp على أنهما متحولان من النوع intptr و realptr على الترتيب ضمن البرنامج السابق .ويمكن تمثيل هذين المؤشرين على الشكل التالي:





و بنا أن هذين المتحولين قد صرح عنهما ضمن البرنامج الرئيسي فإن هذين المتحولين أصبحا متحولين عامين و سيخزنان ضمن مقطع المعطيات لذلك فإن حيز التخزين لمتحولات المؤشرات ip و rp سوف يُخصص بشكل ساكن static أما حيز التخزين لغايات هذه المؤشرات هي متحولات ساكنة أما غايات المؤشرات فهي متحولات ديناميكية.

تخصيص مساحات تخزين ديناميكية allocating dynamic storage:

يخصص استدعاء الإجراء مسبق التعريف NEW ضمن البرنامج السابق مساحة تخزين أثناء تنفيذ البرنامج و ذلك من خلال العبارة التالية:

```
New(rp);
```

يجب أن يكون المتحول الوسيطي الممر إلى new متحولاً من النوع مؤشر.ونوع المعطيات الأساسي لغاية المؤشر سوف يحدد حجم مساحة الذاكرة الواجب تخصيصها . و كل مساحة مخصصة يجب أن توضع في الكومة وفق الخطوات التالية:

١. يطلب البرنامج من مدير الكومة heap manger مساحة متعاقبة من الذاكرة فعلياً بطول ستة بايتات في مثالنا لتخزين عدد حقيقي.
٢. يبحث مدير الكومة عن هذه المساحة و بعد أن يجددها يعطيها للبرنامج و يحذف هذه المساحة من قائمة مواضع الذاكرة المخصصة للكومة التي ينشئها مدير الكومة.
٣. يخزن عنوان هذه المساحة من الذاكرة في المتحول rp أي أن rp سوف يملك قيمة عنوان عند تنفيذ الإجراء new.

تلاحظ عند تنفيذ البرنامج السابق أن الكومة قد نقصت بمقدار ثماني بايتات بعد استدعاء الإجراء new و ذلك لأن مدير الكومة يُقسم الكومة إلى مساحات أطوالها من مضاعفات العدد 8 لذلك و ضمن الاستدعاء السابق حذف من الكومة ثماني بايتات بعد استدعاء الإجراء new و استهلك منها ستة بايتات و بقي بايتين زائدين . و كذلك سوف تنقص الكومة بمقدار ثماني بايتات عند استدعاء الإجراء new من أجل المؤشر ip .

إذا لم يستطع مدير الكومة تخصيص مساحة من المطلوبة فإن البرنامج سوف يتوقف مع خطأ في التنفيذ و من أجل تجنب هذه المشكلة علينا فحص كمية الذاكرة المتبقية في الكومة قبل تخصيص أي كمية من الذاكرة .

مثال:

```
Type realarray=array[1..500] of real;
Raptr=^realarray;
Var rap:raptr;
If maxavail>sizeof(realarray)then
……………………….
```

حتى نستطيع تخصيص مساحة من الذاكرة لمتحول ديناميكي جديد يجب أن تكون المساحة المتبقية في الكومة أكبر أو تساوي المساحة المتبقية أكبر أو تساوي المساحة المطلوبة و في المثال السابق يجب أن تكون أكبر من المساحة المطلوبة لتخزين realarray .

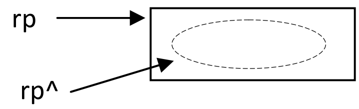
إعداد المهندس خالد ياسين الشيخ
khaledyassinkh@gmail.com

متحول غاية المؤشر accessing target variable:

تبرز العبارة التالية كيفية الإشارة إلى متحول غاية المؤشر:

Rsize:=sizeof(rp^);

يمثل المتحول rp^ متحول غاية المؤشر target variable و الذي عنوانه مخزن في المؤشر rp . و بعارة أدق يمكن القول بأن المتحول rp^ يعني المتحول الذي يشير إليه المؤشر rp كما في الشكل التالي:



و كما لاحظنا فإن متحول غاية المؤشر لا يملك اسماً مستقلاً و لكن يُشار إليه بواسطة اسم مؤشره الخاص تدعى عملية الإشارة إلى هذا المتحول بالإشارة غير المباشرة indirect reference على عكس الإشارة المباشرة direct reference للمتحولات و التي تتم بتحديد اسم المتحول مباشرة.

إن عبارة الإلحاق السابقة تقوم بما يلي:

١. تحديد كمية الذاكرة المخصصة للمتحول rp^ .
٢. إلحاق هذه القيمة بالمتحول rsize.

إلحاق قيم بمتحول غاية المؤشر assigning values to target variable:

يرينا البرنامج التالي كيفية إلحاق قيم بمتحول غاية المؤشر:

code

```
program test;
type
intptr=^integer;
realptr=^real;
var ip:intptr;
rp:realptr;
begin
randomize;
writeln('memavail = ',memavail:10,' bytes');
new(rp);
new(ip);
repeat
rp^:=random;
ip^:=random(500);
writeln('rp^ = ', rp^:10:5,'; ip^ = ',ip^:5);
until (rp^<0.5) and(ip^<250);
writeln('memavail = ',memavail:10,' bytes');
readln
end.
```

خرج البرنامج شبيه بالتالي:

```
memavail =    537840 bytes
rp^ =   0.96095; ip^ =   387
rp^ =   0.67154; ip^ =   459
rp^ =   0.48835; ip^ =   393
rp^ =   0.59126; ip^ =   423
rp^ =   0.75046; ip^ =    43
rp^ =   0.43047; ip^ =   340
rp^ =   0.85400; ip^ =   485
rp^ =   0.22162; ip^ =    60
memavail =    537824 bytes
```

نلاحظ عبارتا الإلحاق في حلقة repeat ضمن البرنامج السابق قيماً عددية بموضعين من الكومة حيث ألحقت قيمة حقيقية بالمتحول ^rp و قيمة صحيحة بالمتحول ^ip و مع ان هاتين القيمتين تنتميان إلى نوعي معطيات مختلفين إلا أن كمية الذاكرة المخصصة لهما و المقتطعة من الكومة متساوية.و ذلك كما قلنا لأن مدير الكومة يُقسم هذه الذاكرة إلى أجزاء من مضاعفات العدد 8 لذلك نلاحظ أن الكومة قد نقصت بمقدار 16 بايتاً.
و كما نلاحظ من خلال البرنامج السابق أن حلقة repeat سوف تتكرر كثر من مرة و لكن حجم الكومة لن ينقص لعدم تخصيص أي حجرات من الذاكرة داخل جسم حلقة repeat و إنما تُلحق القيم بالحجرة التي جرى تخصيصها خارج حلقة .repeat

التمييز بين المؤشر و غاية المؤشر distinction between pointer and target:

ينبغي علينا أولاً و قبل التعامل مع المؤشرات معرفة طريقة تخزين المعلومات باستخدامها إذ يُلحق بمتحول المؤشر عنوان أي لا يمكننا إلحاق قيمة بالمتحول rp ضمن البرنامج السابق و بالتالي العبارة التالية غير صحيحة:

```
rp:=25.5;
```

و ذلك لأن المتحول rp لا ينتمي إلى نوع المعطيات عدد حقيقي و إنما مؤشر يشير إلى عدد حقيقي .
و لذلك الأمر بالنسبة لمتحول غاية المؤشر و الذي دائماً يحتوي على قيمة تنتمي لنوع معطيات ما فلا يمكننا تخصيص ذاكرة له باستخدام الإجراء new أي العبارة التالية أيضاً غير صحيحة:

```
New(rp^);
```

و ذلك لأن الإجراء new يأخذ متحولاً وسيطياً عبارة عن مؤشر في حين أن ^rp متحول من النوع عدد حقيقي.إذا عندما نعرف نوع معطيات مؤشر و نصرح عن متحول ينتمي إليه فإن هذا المتحول يدعى بمتحول المؤشر و يحتوي هذا المؤشر دائماً على عنوان موضع في الذاكرة و يحدد حجمه نوع المعطيات الأساسي للمؤشر يسمى هذا الموضع غاية المؤشر pointer traget (أي المكان الذي يشير إليه المؤشر) و غاية المؤشر كما ذكرنا عبارة عن تتابع حجرات من الذاكرة توضع فيها القيمة امراد تخزينها فإذا أردنا تخزين عنوان غاية المؤشر عندئذ يتم إلحاقها بمتحول المؤشر مباشرة أما إذا أردنا تخزين قيمة للمؤشر عندئذ تُلحق بمتحول غاية المؤشر أي بمتحول المؤشر متبوعاً الرمز ^

ويمكن اختصار الكلام السابق بما يلي:

المؤشرات pointers هي عبارة عن متحولات تحوي عناوين في الذاكرة كقيم تكون مخزنة ضمنها حيث أن المؤشر يتضمن عنواناً لمتحول يحتوي على قيمة معينة حيث أن المتحول يدل بشكل مباشر على قيمة معينة و يدل المؤشر بشكل غير مباشر على قيمة و تسمى عملية الدلالة من خلال المؤشر بالعملية غير المباشرة indirect .
مثل بقية المتحولات يجب التصريح عن المؤشرات قبل استخدامها ويتم ذلك بلغة باسكال من خلال علامة الإدراج ^ caret.

إلحاق المؤشرات pointer assignments:

يمكننا إلحاق محتويات متحول مؤشر إلى متحول مؤشر آخر إذا كان كلا المتحولين ينتمي إلى نوع معطيات واحد و بعبارة أدق يمكن أن يشير المؤشران إلى قيم من نفس النوع .
يرينا البرنامج التالي كيفية إلحاق محتويات متحول مؤشر بمتحول مؤشر آخر:

code

```
program test;
const ff='c:\f.txt';
var f:text;
type
intptr=^integer;
realptr=^real;
var ip,ip2:intptr;
rp,rp2:realptr;
begin
writeln('memavail = ',memavail :10,' bytes');
new(rp);
rp^:=23.3;
rp2:=rp;
writeln('rp^ = ',rp^:10:5,'; rp2^ = ',rp2^:10:5);
new(ip);
ip^:=23;
ip2:=ip;
writeln('ip^ = ',ip^:5,'; ip^= ',ip^:5);
writeln('memavail = ',memavail :10,' bytes');
readln
end.
```

خرج البرنامج كالتالي:

```
memavail =     538000 bytes
rp^ =   23.30000; rp2^ =   23.30000
ip^ =    23; ip^=    23
memavail =     537984 bytes
```

لقد ذكرنا سابقاً أم متحولات المؤشرات تخزن ضمن مقطع المعطيات أو مقطع المكدس و يحدد ذلك كونها متحولات عامة global variables أو متحولات موضعية local variables في حين أن متحولات غايات المؤشرات تخزن في الكومة heap لذلك و في البرنامج السابق لدينا أربعة مؤشرات و لكن لدينا غايتين فقط استهلكتا من الكومة ذاكرة بمقدار 16 بايت . عندما ألحقنا قيمة متحول المؤشر rp بمتحول المؤشر rp2 أصبح لدينا مؤشران يشيران إلى موضع واحد في الذاكرة. حيث تعرف هذه العملية بالتسمية المضاعفة أو البديلة aliasing و هي تسبب بعض المشاكل أحياناً أن لم نكن حريصين بالقدر الكافي.فإذا سُمي موضع ما في الذاكرة بعدة أسماء فإن البرنامج سوف يستطيع تغيير القيمة المخزنة في هذا الموضع بعدة طرق و هذا التنوع سيشكل صعوبة في تتبع خطوات البرنامج مما يؤدي إلى صعوبة في اكتشاف الأخطاء.
نفذ البرنامج التالي و أدخل قيماً كبيرة و من ثم أدخل قيماً صغير بين 0 و 1.

code

```
program test;
const ff='c:\f.txt';
var f:text;
type
intptr=^integer;
realptr=^real;
```

```
var
rp,rp2:realptr;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
procedure transform(var theptr:realptr);
const multfactor=1000.0;
begin
if(theptr^>=0)and (theptr^<1.0)then
theptr^:=multfactor*theptr^
else
theptr^:=1/theptr^;
end;
begin
new(rp);
repeat
getreal('value? (<0 to stop)',rp^);
rp2:=rp;
writeln('rp^ = ',rp^:10:5);
transform(rp2);
writeln('after transform: ');
writeln('rp^ = ',rp^:10:5);
until(rp^<0);
readln
end.
```

يحول البرنامج السابق القيم الكبيرة إلى قيم صغيرة و العكس بالعكس .فعندما يدخل المستخدم قيمة ما فإن البرنامج سوف يخزنها في المتحول ^rp الذي يستخدم فقط عند استدعاء الإجراء getreal و ضمن استدعاءين للإجراء writeln . وبعد أن ألحق البرنامج قيمة بالمتحول rp من خلال استدعاء الإجراء new ألحق قيمة بالمتحول rp2 أي جعل المؤشر الثاني يشير إلى نفس موضع الذاكرة الذي يشير إليه المؤشر الأول أو بمعنى جعل كلا المتحولين rp و rp2 يحتوي على نفس العنوان.

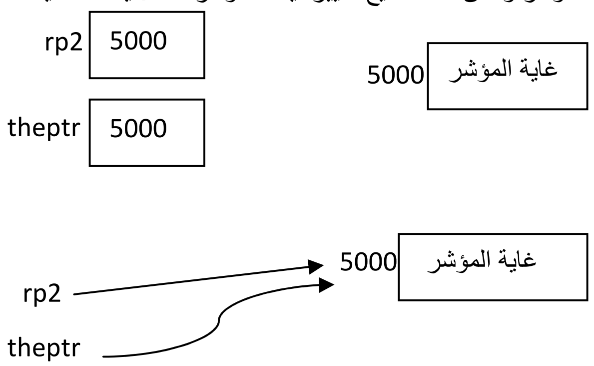
أما استدعاء الإجراء transform فهو يتعامل مع rp2 حيث يعمل على تغيير قيمة المتحول ^theptr و الذي يكافئ تماماً المتحول ^rp2 . و لكن بما rp2 و rp يشيران إلى نفس الموضع من الذاكرة فإن تغيير غاية المؤشر rp2 هي فعلياً تغيير غاية المؤشر rp و ذلك لأن ^rp و ^rp2 هما اسمان لمتحول واحد.

الوسيط الوحيد في الإجراء transform سُبق بالمميز var وكن الإجراء فعلياً لا يغير قيمة المتحول rp2 و إنما يغير قيمة المتحول ^rp2 لذلك فإن حذف المميز var من أمام الوسيط theptr لا يؤثر على تنفيذ البرنامج احذف المميز var و نفذ البرنامج ثانية.

من أجل فهم أكبر للمؤشرات لنفترض أن عناوين حجر الذاكرة هي أعداد من النوع الصحيح عندئذ سوف نخزن في متحول المؤشر قيمة تتألف من بايتين :الأول منها يحتوي على عنوان مقطع segment المخزن فيه غاية المؤشر و الثاني يمثل إزاحة offset غاية المؤشر عن بداية المقطع . فإذا كان لدينا العنوان 5000 مخزن ضمن المتحول rp2 وأيضا ضمن المتحول rp و استدعينا الإجراء transform ومررنا المتحول rp2 بقيمته فإن القيمة المحتواة في rp2 سوف تنسخ و تمرر إلى الإجراء و سوف يفسر الإجراء هذه القيمة (أعني 5000) على أنها عنوان في الذاكرة و سيفهم المتحول ^theptr على أن القيمة المخزنة في العنوان 5000 فإذا تغيرت هذه القيمة فإن هذا التغيير سوف يحفظ و يبقى حتى بعد انتهاء تنفيذ

الإجراء أم القيمة المخزنة في المتحول rp2 فلا يمكن الوصول إليها على الرغم من ارتباط هذه القيمة بغاية المؤشر. لذلك إذا مررنا مؤشراً بقيمته عندئذ يمكننا تغيير غاية المؤشر و لكن لا نستطيع تغيير قيمة المؤشر نفسه: يمكننا تخيل ذلك بالذاكرة:

الرسم البياني:



إلحاق العناوين بالمؤشرات assigning address to pointers:

لقد بينا أن عملية تهيئة المؤشرات تتم باستدعاء الإجراء new أو بواسطة إلحاق متحول مؤشر إلى متحول مؤشر آخر من نفس النوع . و لكن لغة تربو باسكال سمحت بإلحاق عنوان متحول معين إلى متحول مؤشر و لو كان هذا المتحول قد عرف على أنه متحول ساكن static.

١. معامل العنوان @ the address operators:

تزودنا لغة تربو باسكال بالمعامل @ و التابع address من أجل التعامل مع العناوين. حيث يقوم معامل العنوان address operator بتوليد موقع متحوله الوسيطي و هذا المتحول الوسيطي يمكن أن يكون متحولاً أو وسيطاً أو حتى روتيناً.
إذ يأخذ المعامل @ متحولاً وسيطياً واحداً يملك موقعاً في الذاكرة و يعيد موقع هذا المتحول الوسيطي. و هذا المعامل أحادي unary operator و يملك الأسبقية الأعلى.

٢. التابع ADDR the ADDR Function:

يأخذ التابع ADDR مميز متحول أو اسم إجراء أو تابع أو بشكل عام اسم موضع في الذاكرة و يعيد موقع هذا المتحول في الذاكرة. وبما أن هذا التابع يعيد عنواناً في الذاكرة لذلك يمكن أن يقال أنه يعيد مؤشراً.و هذه القيمة التي يعيدها هذا التابع متوافقة مع كل المؤشرات و يمكن إلحاقها بأي متحول مؤشر.
البرنامج التالي يظهر لنا كيفية استخدام المعامل @ و التابع ADDR للحصول على عناوين في الذاكرة:

code
```
program test;
const ff='c:\f.txt';
var f:text;
type
realptr=^real;
var rval1,rval2:real;
rp1,rp2:realptr;
begin
rval1:=100.0;
rval2:=200.0;
rp1:=@rval1;
rp2:=addr(rval1);
if rp1=rp2 then
writeln('pointers contain the same address')
else
```

```
writeln('pointers contain different addresses');
writeln('rp1^ = ',rp1^:10:5,'; rp2^ = ',rp2^:10:5);
readln
end.
```

خرج البرنامج هو:

```
pointers contain the same address
rp1^ =  100.00000; rp2^ =  100.00000
```

يقوم المعامل @ و التابع addr بنفس العمل لذلك لا فرق في استخدام أحدهما مكان الآخر.

استخدام الإجراء Getmem لتخصيص مساحات تخزين ديناميكية

using getmem to allocate dynamic storage :

بالإضافة إلى الإجراء new يوجد في لغة تربو باسكال إجراء آخر لتخصيص مساحات تخزين ديناميكية هذا الإجراء هو getmem و هو يأخذ متحولين وسيطين هما:

١. متحول مؤشر.
٢. متحول من النوع word يمثل عدد البايتات الواجب تخصيصها من الذاكرة ديناميكياً.

فعلى سبيل المثال:تخصص العبارة التالية ستة بايتات من الذاكرة و تلحق هذه البايتات الستة بالمتحول ptrvar:

GETMEM(ptrvar,sizeof(real));

يخصص مدير الكومة heap manager ثمانية بايتات كما ذكرت لأنها أقل وحدة تخصيص يسمح مدير الكومة لبرنامج باستخدامها . لذلك يقع على عاتق المبرمج مسؤولية تخصيص مساحات كافية لتخزين المعلومات التي تريد.

أخطاء المؤشرات pointer pitfalls:

تعتبر المؤشرات أدوات برمجية ممتازة و لكن قوتها هذه لها ثمن يجب أن يُدفع إن لم نمتلك الحرص الكافي الذي يجنبنا الوقوع في معضلات كبيرة قد لا تحل أبداً.لذلك سوف نتناول في الفقرات التالية بعض هذه المشاكل التي تواجهنا أثناء استخدام المؤشرات.

إعادة استخدام المؤشر reusing a pointer:

لقد رأينا أن المؤشرات تأخذ غاياتها من كومة البرنامج من خلال التخصيص الديناميكي لذاكرة الكومة heap و لكن ماذا يحدث إذ أعدنا تخصيص مؤشر تم تخصيص مساحة له؟؟؟؟؟؟؟؟؟ ☎☠

يرينا البرنامج التالي ماذا يحدث إذا استدعينا الإجراء new عدة مرات مع المؤشر نفسه في كل مرة:

code

```
program test;
const ff='c:\f.txt';
var f:text;
const maxtimes=20;
type realptr=^real;
var rp:realptr;
count,firstmem,lastmem:integer;
procedure getval(var theptr:realptr;count:integer);
begin
new(theptr);
theptr^:=random;
writeln(count:2,': theptr^ = ',theptr^:10:5);
```

```
end;
begin
randomize;
firstmem:=memavail;
writeln('memavail = ',memavail:10,' bytes');
for count:=1 to maxtimes do
getval(rp,count);
writeln('      rp^   = ',rp^:10:5);
lastmem:=memavail;
writeln('memavail = ',memavail:10,' bytes');
writeln('the heap was decrease',firstmem-lastmem,' bytes');
readln;
end.
```

خرج البرنامج شبيه:

```
memavail =     537776 bytes
 1: theptr^ =    0.57164
 2: theptr^ =    0.76601
 3: theptr^ =    0.63058
 4: theptr^ =    0.35267
 5: theptr^ =    0.88830
 6: theptr^ =    0.08712
 7: theptr^ =    0.40856
 8: theptr^ =    0.52175
 9: theptr^ =    0.21448
10: theptr^ =    0.54680
11: theptr^ =    0.97480
12: theptr^ =    0.31229
13: theptr^ =    0.06454
14: theptr^ =    0.78690
15: theptr^ =    0.63893
16: theptr^ =    0.52420
17: theptr^ =    0.44659
18: theptr^ =    0.64069
19: theptr^ =    0.28085
20: theptr^ =    0.87507
          rp^   =    0.87507
memavail =     537616 bytes
the heap was decrease160 bytes
```

يستدعي البرنامج السابق الإجراء getval عدة مرات و في كل مرة يخصص الإجراء مساحة لتخزين عدد حقيقي و يجعل المؤشر theptr يؤشر إلى هذا المتحول (مع ملاحظة أن المتحول theptr يوافق المتحول العام rp) وبعد انتهاء هذه الاستدعاءات المتكررة يفحص البرنامج الحجم المتبقي من ذاكرة الكومة.
نلاحظ من خلال البرنامج السابق أن البرنامج قد أنقص ذاكرة الكومة بمقدار 160 بايتاً في حيت لدينا متحولاً واحداً هو rp^ أما المتحولات التسعة عشر التي أنشاها فقد فقدت؟ ففي كل استدعاء للإجراء new جرى تغيير محتويات theptr بكتابة عنوان جديد فيه و بذلك يفقد العنوان القديم و تفقد معه مقدرة لبرنامج على الوصول إلى القيم المخزنة في ذلك العنوان(توجد طريقة لتجاوز هذه العقبة و هي إنشاء أنواع معطيات مركبة و قوية تدعى باللوائح lists).

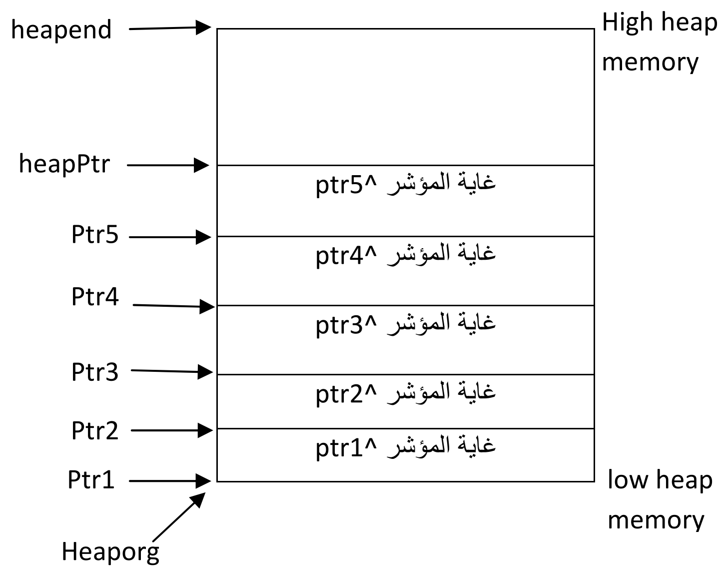
إعادة مساحة من الذاكرة إلى الكومة returning storage to the heap:

إعادة إرجاع مساحة بعض حجرات من الذاكرة مخصصة لمتحول ما إلى ذاكرة الكومة و هذه العملية تعرف بإنهاء التخصيص reallocation للذاكرة و إنهاء التخصيص هذا يعيد مساحات الذاكرة المخصصة و يجعلها قابلة للاستخدام من جديد و للقيام بذلك نستخدم الإجراء dispose الذي يمكننا من إعادة حجرات من الذاكرة خصصت بواسطة الإجراء new . يأخذ هذا الإجراء متحولا وسيطياً واحداً عبارة عن مؤشر و يعيد المساحة المخصصة لمتحول غاية المؤشر pointer target variable إلى الكومة.وسأضرب مثالاً لتوضيح العملية:

لنفرض أن لدينا الاستدعاءات الخمسة التالية للإجراء new:

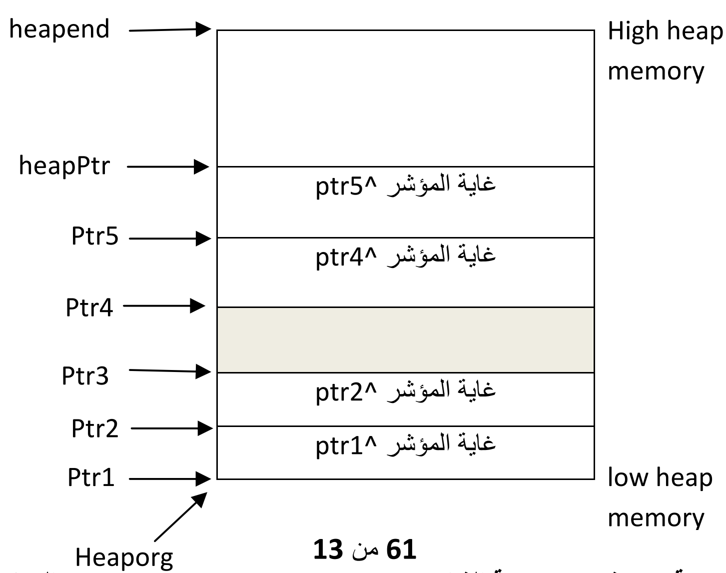
```
New(ptr1);
New(ptr2);
New(ptr3);
New(ptr4);
New(ptr5);
```

عندئذ ستخصص مساحات من الذاكرة لغايات هذه المؤشرات كما في الشكل:

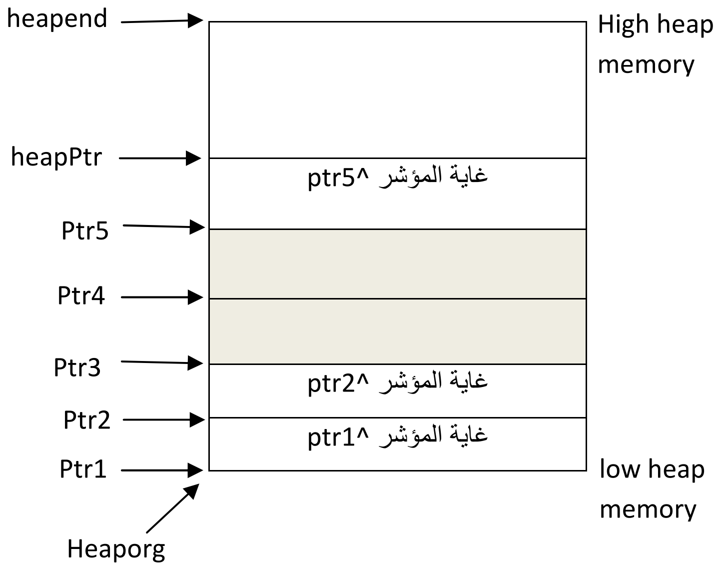


حيث يشير المؤشر مسبق التعريف heapptr إلى قمة الكومة دائماً لذلك تتغير قيمة هذا المؤشر بعد كل عملية تخصيص للذاكرة ليشير إلى قمة الكومة.

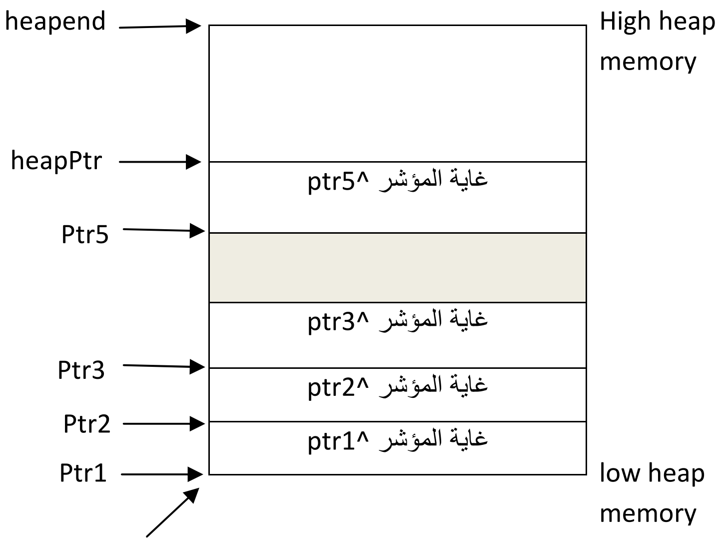
فإذا استدعينا الآن الإجراء dispose للمؤشر ptr3 عندئذ سوف ينهي تخصيص غاية المؤشر ptr3 من الكومة كما هو ممثل في الشكل التالي:



و من أجل استدعاء آخر للإجراء dispose مع المؤشر ptr4 عندئذ ستصبح الكومة كما يلي:



نلاحظ من خلال الشكلين السابقين أن عملية إنهاء التخصيص تلك قد تركت فجوة فارغة hole ضمن المساحة المستخدمة من الكومة و عند تخصيص مؤشر جديد يجري فحص هذه المساحة من حيث كونها تكفي لتخزين غاية المؤشر الجديد. فإذا كانت تكفي عندئذ يخصص هذا الأمر في هذه المساحة و إلا فإنها تترك فارغة و يخصص هذا المؤشر في مساحة فارغة أخرى.
فإذا أعدنا مثلاً تخصيص المؤشر ptr3 مرة أخرى فسوف يتوضع في ذاكرة الكومة على الشكل التالي إن لم تشغل هذه المساحة بغاية مؤشر آخر:



يستخدم البرنامج التالي الإجراء dispose لإنهاء تخصيص المؤشرات .
نفذ البرنامج التالي و قارن خرجه مع البرنامج السابق:

```
code
program test;
const ff='c:\f.txt';
var f:text;
const maxtimes=20;
type realptr=^real;
var rp:realptr;
```

```
count,firstmem,lastmem:integer;
procedure getval( theptr:realptr;count:integer);
begin
dispose(theptr);
new(theptr);
theptr^:=random;
writeln(count:2,': theptr^ = ',theptr^:10:5);
end;
begin
randomize;
firstmem:=memavail;
writeln('memavail = ',memavail:10,' bytes');
new(rp);
for count:=1 to maxtimes do
getval(rp,count);
writeln('     rp^   = ',rp^:10:5);
dispose(rp);
lastmem:=memavail;
writeln('memavail = ',memavail:10,' bytes');
writeln('the heap was decrease',firstmem-lastmem,' bytes');
readln;
end.
```

خرج البرنامج شبيه:

```
memavail =     538048 bytes
 1: theptr^ =    0.82417
 2: theptr^ =    0.77092
 3: theptr^ =    0.14624
 4: theptr^ =    0.78537
 5: theptr^ =    0.81849
 6: theptr^ =    0.54463
 7: theptr^ =    0.09461
 8: theptr^ =    0.27311
 9: theptr^ =    0.71989
10: theptr^ =    0.03222
11: theptr^ =    0.77043
12: theptr^ =    0.51637
13: theptr^ =    0.63104
14: theptr^ =    0.93233
15: theptr^ =    0.34339
16: theptr^ =    0.49618
17: theptr^ =    0.86993
18: theptr^ =    0.64917
19: theptr^ =    0.65563
20: theptr^ =    0.27853
          rp^   =    0.27853
memavail =     538048 bytes
the heap was decrease0 bytes
```

الفرق الرئيسي بين البرنامجين السابقين هو أن هذا البرنامج استخدم من الكومة مساحة مقدارها 0 في حين ان البرنامج السابق قد استخدم 160 بايتاً و ذلك لأن متحولات غايات المؤشرات كانت تعاد إلى الكومة قبل تخصيص مساحة جديدة للمتحول theptr يسبب الإجراء dispose خطأ في تنفيذ البرنامج إذا مرر إليه متحول وسيطي من النوع مؤشر لم تخصص له أي مساحة تخزينية من الذاكرة بعد.لذلك استدعاء الإجراء new يجنباً مثل هذه الأخطاء و كخلاصة يمكنني أن أقول:
يستخدم الإجراء dispose لإنهاء تخصيص مؤشرات قد تم تخصيص قيم لها بواسطة الإجراء new.

استخدام الإجراء freemem لإعادة مساحات الذاكرة المخصصة:

Using freemem to return allocated storage

يستخدم الإجراء freemem لإعادة حجرات من الذاكرة المخصصة بواسطة الإجراء GETMEM حيث يأخذ هذا الإجراء متولين وسيطين هما:

١. متحول مؤشر.
٢. متحول من النوع word يمثل عدد البايتات التي سُتعاد إلى الكومة.

يظهر لنا البرنامج التالي كيفية استخدام الإجراء freemem:

```
code
program test;
const ff='c:\f.txt';
var f:text;
type
realarray=array[1..500] of real;
raptr=^realarray;
realptr=^real;
longintptr=^longint;
var ra1:raptr;
rp1:realptr;
lp1:longintptr;
begin
write('memavail = ',memavail:10,' bytes');
writeln('; maxavail = ',maxavail:10,' bytes');
getmem(ra1,sizeof(realarray));
getmem(rp1,sizeof(real));
getmem(lp1,sizeof(longint));
writeln('after allocating ra1^,rp1^,lp1^');
write('memavail = ',memavail:10,' bytes');
writeln('; maxavail = ',maxavail:10,' bytes');
writeln('------------------------------------');
 freemem(ra1,sizeof(realarray));
 freemem(rp1,sizeof(real));
 freemem(lp1,sizeof(longint));
 writeln('after freeing ra1^,rp1^,lp1^');
 write('memavail = ',memavail:10,' bytes');
writeln('; maxavail = ',maxavail:10,' bytes');
 readln;
 end.
```

خرج البرنامج شبيه بالتالي:

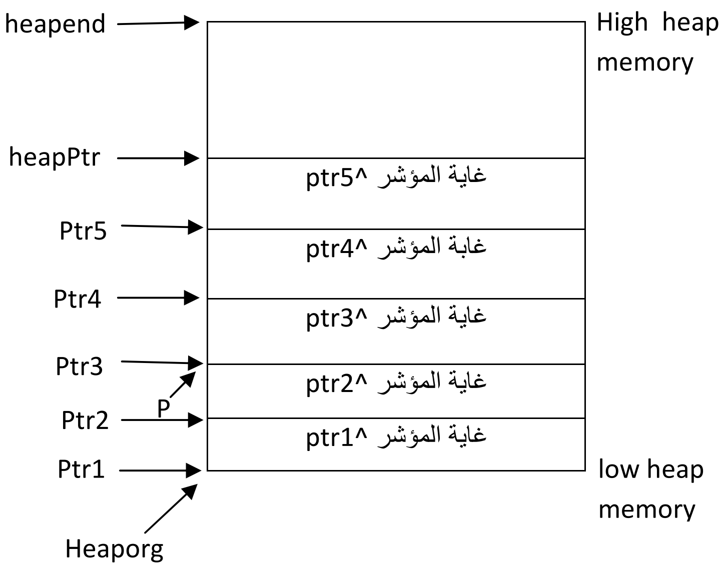
```
memavail =    539568 bytes; maxavail =    539568 bytes
after allocating ra1^,rp1^,lp1^
memavail =    536552 bytes; maxavail =    536552 bytes
------------------------------------
after freeing ra1^,rp1^,lp1^
memavail =    539568 bytes; maxavail =    539568 bytes
```

يجب أن لا ننسى أبداً إن الإجراء freemem ينهي تخصيص حجرات من الذاكرة خصصت بواسطة الإجراء getmem . و عدد هذه الحجرات يجب أن يساوي تماماً عدد الحجرات المخصصة بواسطة الإجراء getmem . و هذا الإجراء يترك فجوات holes بنفس الطريقة التي يتركها الإجراء dispose و تعالج هذه الفراغات بنفس الطريقة التي عولجت بها سابقاً.

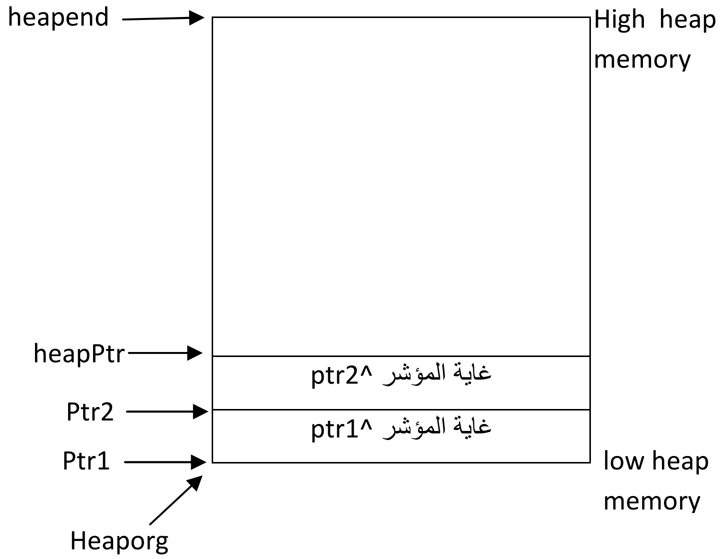
إنهاء تخصيص مجموعة مؤشرات من الكومة:

Reallocating groups on the heap:

لقد استخدمنا حتى الآن الإجراءين dispose و freemem من أجل إنهاء تخصيص حجر من الذاكرة قد خصصت بواسطة الإجراءين new و GETMEM . و هذان الإجراءان يتركان فجوات ضمن ذاكرة الكومة عند إنهاء تخصيص مؤشر ما . لكننا يمكننا استخدام الإجراءين mark و release لإنهاء تخصيص مجموعة من المؤشرات دفعة واحدة و ذلك بإلحاق القيمة الحالية لمؤشر قمة الكومة heapPtr بالمتحول الوسيطي الممرر للإجراء mark و من ثم استدعاء الإجراء release (بعد عدد من الخطوات) مع نفس المتحول الوسيطي السابق عندئذ سيقوم الإجراء بإنهاء تخصيص كل المؤشرات التي خصصت بعد استدعاء الإجراء mark.و كمثال على هذين الإجراءين ليكن لدينا تسلسل استدعاءات الإجراءات التالية:



نلاحظ أن شكل الكومة يشبه هنا يشبه تماماً شكلها الذي رسمناه في الفقرات السابقة بدون استدعاء الإجراء mark و لكن الفرق هنا أن المؤشر p قد أصبح يشير إلى قمة الكومة بعد أول استدعاءين . فإذا استدعينا الآن لإجراء release مع المتحول الوسيطي p فسيجري إنهاء تخصيص جميع المؤشرات التي خصصت بعد استدعاء الإجراء mark أي في مثالنا المؤشرات ptr3 و ptr4 و ptr5 كما هو موضح في الشكل التالي:



نلاحظ من هذا التمثيل لذاكرة الكومة أن الإجراء released لا يترك فجوات فارغة ضمن الكومة.و أخيراً استدعاء الإجراء released مع المؤشر heaporg سوف ينهي تخصيص جميع المؤشرات الموجودة في الكومة لأن هذا المؤشر مسبق التعريف و هو يشير دائما إلى أسفل الكومة أي يشير إلى أول بايت في الكومة و لذلك فإن استدعاء الإجراء realease مع هذا المؤشر سوف يحذف كل المؤشرات من الذاكرة.

القيمة NIL " لا شيء" مسبقة التعريف الخاصة بالمؤشر:

A predefined nil value for pointer:

هناك طريقة أخرى تجنبنا الخطأ الذي سيحدث إذا استدعين الإجراء dispose مع متحول وسيطي (أي مؤشر) غير معرف إذ يمكننا استخدام عنوان خاص هو اللاشيء NIL. حيث أن nil قيمة مسبقة التعريف تستخدم مع المؤشرات و تمثل موضعاً خاصاً في الذاكرة يُلحق بالمؤشر لإعطائه قيمة داخلية و سأزيد الأمر إيضاحاً فلمؤشر الذي يملك القيمة nil لا يملك غاية target و لكن في الحقيقة تمت تهيئة المؤشر و أخذ قيمة ما.

يقوم البرنامج التالي بنفس العمل الذي يقوم به البرنامج السابق إلا إن هذا البرنامج لم يستدع الإجراء new ضمن البرنامج الرئيسي و لكنه حمل المؤشر rp بالقيمة nil.

code

```
program test;
const ff='c:\f.txt';
var f:text;
const maxtimes=20;
type realptr=^real;
var rp:realptr;
count:integer;
procedure getval(var theptr:realptr;count:integer);
begin
if theptr<>nil then
dispose(theptr);
new(theptr);
theptr^:=random;
writeln(count:2,': theptr^ = ',theptr^:10:5);
```

```
end;
begin
randomize;
writeln('memavail = ',memavail:10,' bytes');
rp:=nil;
for count:=1 to maxtimes do
getval(rp,count);
writeln('     rp^ =    ',rp^:10:5);
dispose(rp);
writeln('memavail = ',memavail:10,' bytes');
readln
end.
```

خرج البرنامج شبيه بالتالي:

```
memavail =    537824 bytes
 1: theptr^ =    0.19589
 2: theptr^ =    0.32687
 3: theptr^ =    0.50814
 4: theptr^ =    0.98438
 5: theptr^ =    0.15352
 6: theptr^ =    0.88266
 7: theptr^ =    0.91343
 8: theptr^ =    0.08280
 9: theptr^ =    0.70779
10: theptr^ =    0.47563
11: theptr^ =    0.54005
12: theptr^ =    0.29341
13: theptr^ =    0.99121
14: theptr^ =    0.25828
15: theptr^ =    0.35054
16: theptr^ =    0.17864
17: theptr^ =    0.31462
18: theptr^ =    0.02879
19: theptr^ =    0.30520
20: theptr^ =    0.13743
     rp^ =     0.13743
memavail =    537824 bytes
```

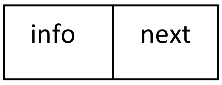
يفحص الإجراء GetVal في البرنامج السابق متحول المؤشر theptr فإذا أعطي هذا المتحول القيمة nil فإن الإجراء لن يستدعي الإجراء dispose.

بنى التأشير الذاتي self referential structures:

تستخدم البرامج عادة عددا من العناصر التي تنتمي لنفس نوع المعطيات و عدد العناصر تلك يُحدد عادة أثناء ترجمة البرنامج Compile و لكن قد نحتاج إلى عدد من العناصر عددها لا نستطيع تحديده أثناء ترجمة البرنامج و إنما أثناء تنفيذه.
فعلى سبيل المثال: لنفرض أننا نريد تحليل أحرف الكلمات المقروءة من ملف ما و لكننا لا نعلم مسبقاً عدد الكلمات التي سوف تقرأ من الملف لذلك سوف نستخدم التخصيص الديناميكي للذاكرة أي لتخصيص مساحات من الذاكرة لكل عنصر مطلوب.
عنصر المعطيات الأساسي المستخدم في مثل هذه البرامج هو نوع معين من السجلات حيث يمكن أن يحوي السجل حقول تنتمي إلى نوع المعطيات مؤشر.
مثال:

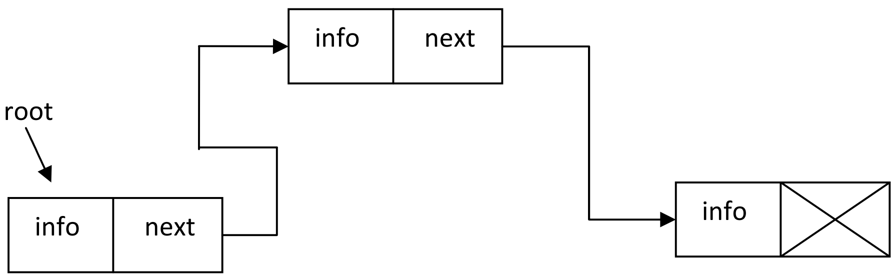
```
Type nodeptr=^node;
node=record
info:real;
next:nodeptr;
end;
var root:nodeptr;
```

تصرح التعريفات السابقة عن نوع معطيات سجل اسمه node يملك حقلين أحدهما يدعى info و ينتمي إلى نوع المعطيات عدد حقيقي و الآخر يدعى next و هو عبارة عن مؤشر يشير إلى عنصر من نفس نوع المعطيات أي يشير إلى سجل آخر من النوع node . لذلك تعتبر البنية node بنية خاصة لأنها تحتوي على معلومات عددية و مؤشر.والشكل التالي يبين لنا طريقة تمثيل حقول هذه البنية:



ربط بنى التأشير الذاتي linking self referential structures:

بما أن كل سجل node يمكن أن يشير إلى سجل node آخر عندئذ يمكننا بناء سلسلة طويلة من هذه البنى و ذلك بربط هذه البنى بعضها البعض بواسطة الحقل next . لذلك يمكننا تعريف اللائحة المترابطة linked list بأنها سلسلة من العناصر رُبط كل عنصر فيها بالعنصر الذي يليه و هكذا....دواليك.
يشار إلى العقدة الأولى في اللائحة المترابطة بمؤشر يشير إلى node و هذا المؤشر هو مؤشر مستقل أعني يجب أن لا يكون المؤشر حقلاً في سجل ما مثلا حقلاً في السجل node .
حيث يُصرح عن متحول الجذر root variable في مقدمة اللائحة المترابطة أما الحقل next الموجود ضمن أول عقدة node فسوف يشير إلى عقدة تالية و هكذا. يمكن أن تحتوي اللائحة على عدد من العقد تكفي لاستخدامات البرنامج أو بقدر ما تبقى في الكومة heap من ذاكرة.
أما الحقل next في العقدة الأخيرة من اللائحة المترابطة يجب أن يشير إلى اللاشيء أي nil حتى يخبرنا بانتهاء عقد اللائحة المترابطة.
الشكل التالي يُرينا لائحة مترابطة تحتوي ثلاثة عناصر. و تجدر الإشارة هنا إلى أن مؤشر الجذر root ليس عقدة مثل node و إنما هو مؤشر عادي يشير إلى العقدة node.



بناء اللوائح المترابطة building linked lists:

اللائحة المترابطة عبارة عن بنية معطيات ديناميكية أي أن حجمها يتحدد فقط عند تنفيذ البرنامج و هذه البنى تختلف عن الأنساق التي يتم تحديد حجمها ونوعها مسبقاً.

يمكننا بناء اللائحة المترابطة عقدة عقدة. فمن أجل إضافة عقدة إلى لائحة مترابطة موجودة نقوم بما يلي:

١. ننشئ عقدة بواسطة استدعاء الإجراء new مع متحول وسيطي مناسب.
٢. نخزن المعلومات المطلوب تخزينها في العقدة.
٣. نربط العقدة الجديدة باللائحة القديمة عبر بعض التعديلات على المؤشرات لجعلها تشير إلى العقدة الجديدة و جعل الحقل next في العقدة الجديدة يشير إلى عقدة معينة في اللائحة المترابطة.

إضافة العقد إلى اللائحة المترابطة adding at the front of a linked list:

يمكننا بناء لائحة مترابطة و ذلك بإضافة كل عقدة جديدة إلى بداية هذه اللائحة. و البرنامج التالي يرينا طريقة القيام بهذه العملية:

code

```
program test;
type
nodeptr=^node;
node=record
info:real;
next:nodeptr;
end;
var temp,root:nodeptr;
procedure getreal(message:string; var value:real);
begin
write(message,' ');
readln(value);
end;
procedure writelist(var thenode:nodeptr);
begin
writeln(thenode^.info:10:5);
if thenode^.next<>nil then
writelist(thenode^.next);
end;
procedure initnode(var thenode:nodeptr);
begin
new(thenode);
thenode^.info:=0.0;
thenode^.next:=nil;
end;
procedure add_front(toadd:nodeptr; var wheretoadd:nodeptr);
begin
toadd^.next:=wheretoadd;
wheretoadd:=toadd;
end;
begin
root:=nil;
temp:=nil;
repeat
initnode(temp);
```

```
getreal('value (<0 to stop) ',temp^.info);
add_front(temp,root);
until temp^.info<0.0;
writelist(root);
readln;
end.
```

إذا نفذنا البرنامج السابق و أدخلنا القيم التالية على الترتيب 6 و 3 و 23 و 25 و 8 و 5 و 9 و 25 و -1 عندئذ يظهر الخرج التالي على الشاشة:

```
value (<0 to stop)  6
value (<0 to stop)  3
value (<0 to stop)  23
value (<0 to stop)  25
value (<0 to stop)  8
value (<0 to stop)  5
value (<0 to stop)  9
value (<0 to stop)  25
value (<0 to stop)  -1
 -1.00000
 25.00000
  9.00000
  5.00000
  8.00000
 25.00000
 23.00000
  3.00000
  6.00000
```

يعرف البرنامج السابق المتحولين root و temp على أنهما مؤشران يشيران إلى السجل node حيث يعمل المؤشر root كدليل داعم للائحة المترابطة التي سوف تبنى.لذلك يشير المؤشر root دائماً إلى أول عقدة في هذه اللائحة حتى ولو تغيرت هذه العقدة أثناء تنفيذ البرنامج و ذلك لأننا نحتاج إلى المؤشر root حتى نصل إلى أي عنصر في اللائحة المترابطة.
يشير المؤشر temp إلى العقدة الجديدة أثناء تهيئتها و قبل إضافتها إلى اللائحة المترابطة إذ يشير temp إلى موضع التخزين المخصص للعقدة الجديدة ومن ثم تضاف هذه العقدة إلى اللائحة المترابطة وعندئذ يشير temp إلى عقدة جديدة و يتخلى عن تأشيره للعقدة السابقة و هكذا....دواليك.

الإجراء initnode:

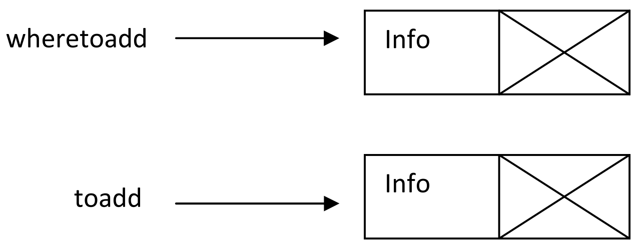
يخصص الإجراء initnode مساحات الذاكرة المطلوبة من أجل عقدة جديدة ويعيد هذا الإجراء مع مساحات الذاكرة تلك مؤشراً يشير إلى هذه المساحات .
و هذا الإجراء يستخدم في تخصيص مساحات الذاكرة الإجراء new لذلك ينطبق عليه ما ينطبق على الإجراء new أعني إذا لم تبق مساحات من الذاكرة تكفي لعقدة جديدة فإن البرنامج لن يستطيع تخصيص مساحة من الذاكرة و سوف يحدث خطأ في تنفيذ البرنامج. و قد تعلمنا كيفية معالجة و تجنب مثل هذه الأخطاء.
ويجب التأكد من أن حقل next في كل عقدة جديدة قد أعطي القيمة nil أي أصبح يشير إلى اللاشيء لأن معظم الروتينات التي تتعامل مع اللوائح المترابطة تبحث عن القيمة nil لتحدد وصولها إلى نهاية اللائحة أم لا فإذا لم تهيأ هذه المؤشرات فإن آخر عقدة لن تحتوي nil وسوف يحاول البرنامج عندئذ قراءة عقدة بعد العقدة الأخيرة و هذا سوف يسبب خطأ في تنفيذ البرنامج.

لاحظ أن صيغة تحديد الحقل الموجود في متحول غاية المؤشر تشبه صيغة التعامل مع السجل و هي : thenode^.node و ليست ^thenod.node و ذلك لأن صيغة تحديد متحولات غايات المؤشرات تستخدم مميز المتحول متبوعاً بالرمز ^ و في مثالنا السابق كان اسم المتحول هو thenode و بعد تحديد متحول غاية المؤشر يمكننا الوصول إلى الحقول المستقلة باستخدام النقطة (.) متبوعة باسم الحقل.

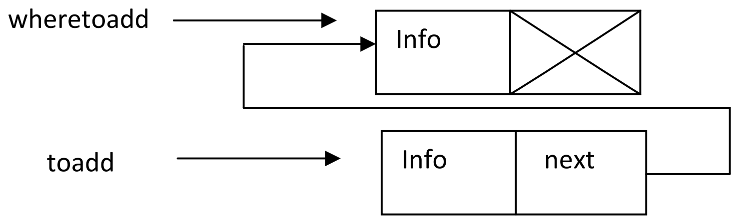
الإجراء add_front :

بعد أن يتم تخصيص مساحة من الذاكرة لعقدة جديدة فإن الحقل info فإن الحقل info في هذه العقدة سوف يُعطى القيمة 0 و من ثم يستدعي الإجراء add_front الذي يضع العقدة الجديدة في بداية اللائحة المترابطة. يظهر الشكل التالي لنا عملية إضافة العقدة الثانية.

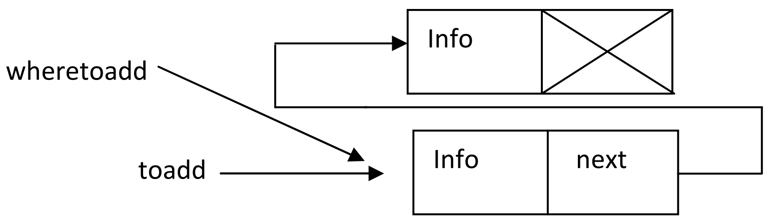
A. at start of procedure



B. after toadd^.next:=wheretoadd



C. after wheretoadd :=toadd;



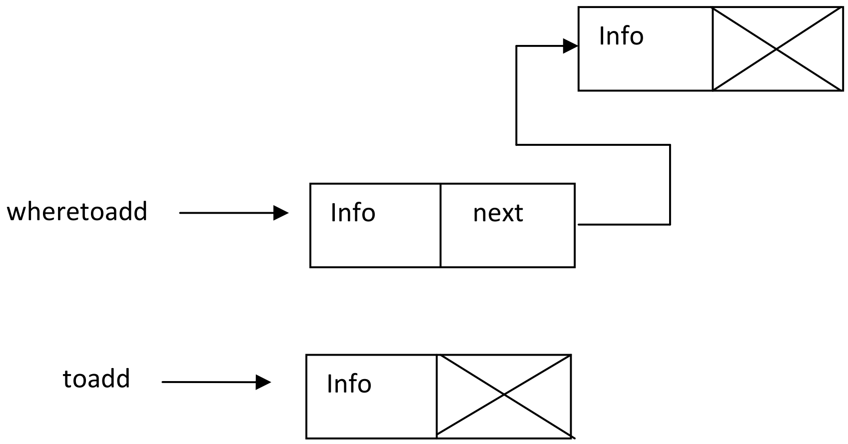
لاحظ أن عبارة واحدة wheretoadd و toadd^.next (أو root و temp^.next على الترتيب ضمن البرنامج الرئيسي)يشيران إلى نفس العقدة.

إن خوارزمية هذا العمل تقوم على جعل الحقل next من المتحول ^toadd تشير إلى العقدة الأولى و من ثم جعل المؤشر wheretoadd و الذي يكافئ root يشير إلى العقدة الجديدة. وترتيب هذه العمليات مهم جداً و يجب أن لا يغيب عن ذاكرتنا أن الطريقة الوحيدة للوصول إلى عقد اللائحة تتم عن طريق مؤشر الجذر أي المؤشر الذي يشير إلى أول عقدة لأن هذا المؤشر هو المتحول الوحيد الذي خصص تخصيصاً ساكناً statically ضمن اللائحة المترابطة و هذا يعني بالضرورة أن هذا المتحول هو الوحيد الذي يمكن الوصول إليه مباشرة.

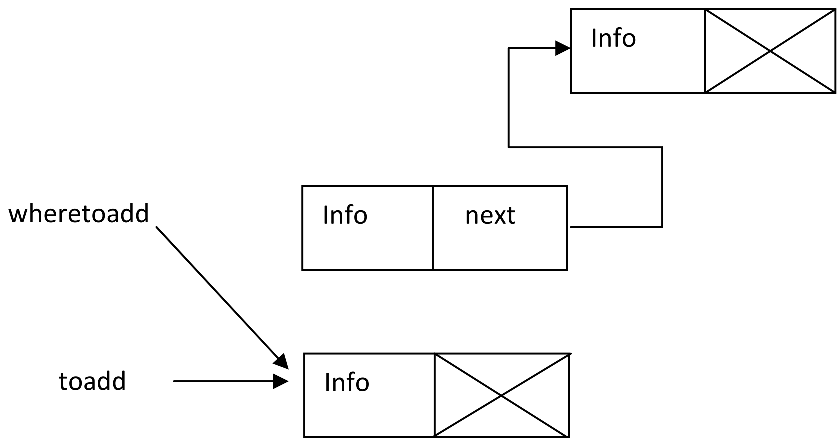
إذا جعلنا المؤشر wheretoadd يشير إلى العقدة الجديدة مباشرة فإننا لن نستطيع الوصول إلى لائحة مفقودة لأن عبارة الإلحاق تلك غيرت الطريقة الوحيدة للوصول إلى العنصر الأول في اللائحة الأصلية ^wheretoadd.

الشكل التالي يرينا كيف تحدث علية الفقدان تلك:

A. at start of procedure



B. original list is lost because wheretoadd no longer pointer to it



الجزء A في هذا الشكل يرينا وضع المؤشرات قبل استدعاء الإجراء add_front و يرينا عقدة جديدة تنتظر إضافتها إلى اللائحة المترابطة و هذه اللائحة كما نلاحظ يتم الوصول إلى عناصرها من خلال المؤشر wheretoadd (أو ما يُدعى بمؤشر الجذر root ضمن البرنامج الرئيسي).
أما في الجزء B فإن دليل اللائحة المترابطة أشار إلى العقدة الجديدة مباشرة و عندئذ فقدنا اللائحة الأصلية لأننا لن نعد نستطيع الوصول إلى العنصر الأول من خلال دليل اللائحة. و حتى نتجن هذه المشكلة و التي قد تسبب ضياعاً كبيراً للمعلومات . علينا أولاً جعل حقل next في العقدة الجديدة يشير إلى بداية اللائحة و عندئذ فقط يمكن فك الربط بين دليل اللائحة و اللائحة نفسها.

الإجراء writelist:

يُظهر هذا الإجراء محتويات العقد المستقلة في اللائحة المترابطة بنفس ترتيبها الموجود في اللائحة المترابطة .
لقد استخدمنا في بناء هذا الإجراء تقنية التعاوبة أو العودية (التراجعية) recursive حيث يظهر هذا الإجراء محتويات الحقل info للعقدة الحالية و من ثم يستدعي نفسه من أجل العقدة التالية في اللائحة حتى يصل هذا الروتين إلى العقدة الأخيرة أي التي يؤشر الحقل next فيها إلى اللاشيء nil. يُستدعى الإجراء مع جذر اللائحة المترابطة أي العقدة الأولى فيظهر هذا الإجراء محتويات الحقل info فيها ومن ثم يفحص الإجراء وصوله إلى نهاية اللائحة فإذا كانت التعبير التالي هي true :

Thenode^.next<>nil

هذا يعني أن الإجراء لم يصل إلى نهاية اللائحة عندئذ يعيد الإجراء استدعاء نفسه مع مؤشر العقدة التالية أي مع مؤشر الحقل next للعقدة الحالية و ذلك باستخدام العبارة التالية:

writelist(thenode^.next);

أما إذا كان ناتج التعبير السابق هو false فهذا يعني وصول الإجراء إلى نهاية اللائحة المترابطة و عندئذ يُنهي الإجراء نفسه.
و بما أن تعليمة إظهار محتويات الحقل info أي استدعاء الإجراء writeln سبقت الاستدعاء العودي فإن محتويات هذه العقد سوف تكتب ابتداء من العقدة الأولى و حتى العقدة الأخيرة.
و لكن إذا غيرنا موضع العبارة writeln في الإجراء السابق ووضعناها بعد الاستدعاء العودي فإن الإجراء writelist سوف يكتب محتويات العقد ابتداءً من العقدة الأخيرة و حتى العقدة الأولى (علل ذلك؟)

إضافة العقد في أي مكان ضمن اللائحة المترابطة adding elsewhere in a linked list:

يمكننا أيضاً بناء لائحة مترابطة بإضافة عناصر جديدة في نهاية هذه اللائحة كما هو موضح في الإجراء التالي :

code

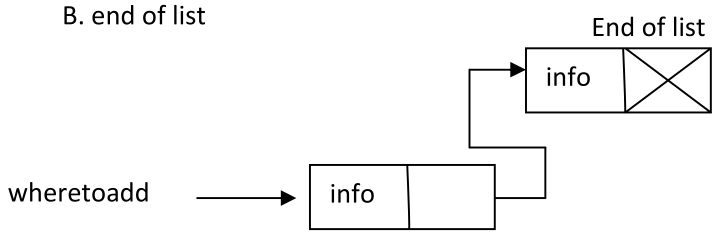
```
procedure add_back(toadd:nodeptr; var wheretoadd:nodeptr);
begin
if wheretoadd=nil then
wheretoadd:=toadd
else
add_back(toadd,wheretoadd^.next);
end;
```

حتى ترى كيف يعمل هذا الإجراء أضف هذا الإجراء إلى قسم للبرنامج السابق ومن ثم استبدل استدعاء الإجراء add_front في البرنامج الرئيسي بالاستدعاء add_back .
يضيف الإجراء السابق عقد جديدة إلى نهاية اللائحة الحالية وخوارزمية عملية الإضافة تلك تعتمد على عملية العبور traverse أي الانتقال ضمن القائمة حتى نصل نهايتها حيث سنجد المؤشر يشير إلى القيمة nil .
الشكل التالي يظهر موضعين يمكن أن نجد فيها القيمة nil ضمن اللائحة المترابطة:

A. empty list

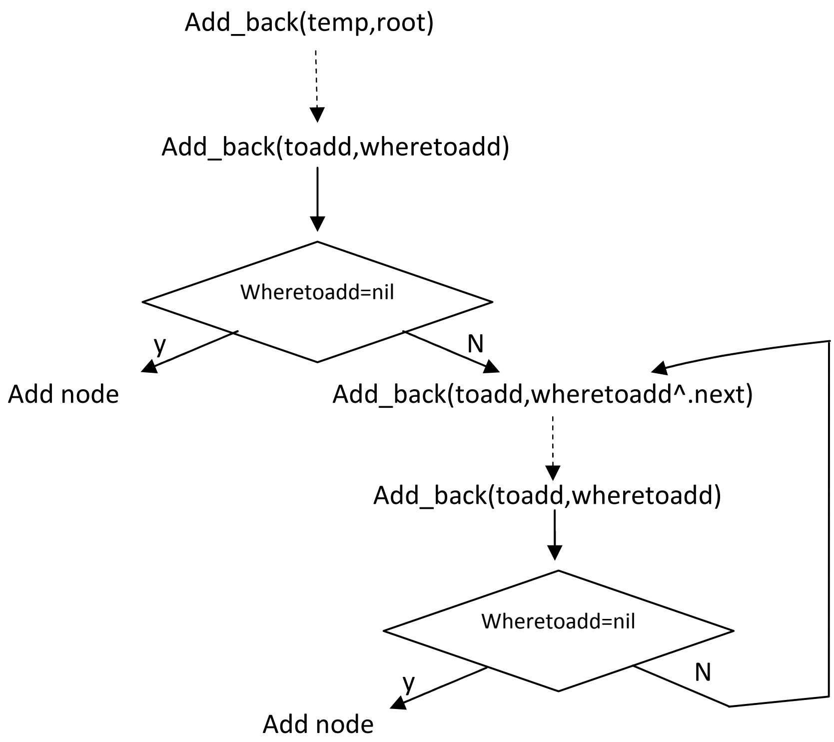
wheretoadd ⟶ NIL

B. end of list



عندما يصل الإجراء add_back إلى مؤشر يشير إلى القيمة nil فإن العقدة الجديدة سوف تُضاف إلى اللائحة و هذه اللائحة المعدلة سوف تعاد إلى الروتين المستدعى . إذا كانت هناك عقدة في الموقع الحالي للمؤشر wheretoadd فإن الإجراء add_back يستدعي نفسه مع المؤشر الموجود في الحقل next لهذه العقدة أي wheretoadd^.next كمتحول وسيطي جديد حتى يصل إلى نهاية اللائحة حيث الاستدعاء الأخير للإجراء add_back اللائحة محتوية على العقدة الجديدة إلى الروتين المستدعي وفق سلسلة من التمريرات و في الجزء الخارجي لهذه السلسلة من التمريرات فإن اللائحة المشار إليها بالمؤشر root سوف تعدل و هو ما نريد تماما (المؤشر root (wheretoadd) سوف يعاد إلى بداية السلسلة من خلال انهيار العودية).

ويبين الشكل التالي المخطط المنطقي للعمليات التي تتم ضمن add_back:



يسير السطر الأول في الشكل السابق إلى نفس الوسطاء المستخدمة في استدعاء الإجراء من قبل الروتين المستدعي أما في السطر الثاني فإن الوسطاء المستخدمة هي الوسطاء التي استخدمها الإجراء نفسه.

ترتيب لائحة مترابطة ordering a linked list:

قد نحتاج في بعض الأحيان إلى ترتيب عناصر اللائحو وفق طريقة ما .
فعلى سبيل المثال قد نرغب في ترتيب عقد اللائحة المترابطة وفق قيمها العددية الموجودة في الحقل info . يبني الإجراء التالي لائحة مرتبة واضعاً العقدة التي قيمة الحقل info فيها أقل من باقي العقد في بداية هذه اللائحة .
حتى ترى كيف يعمل هذا الإجراء أضفه للبرنامج السابق و أجر التعديل المطلوب وأعني استبدل استدعاء الإجراء add_back باستدعاء الإجراء add_start.

code

```
procedure add_start(toadd:nodeptr; var wheretoadd:nodeptr);
begin
if wheretoadd=nil then
wheretoadd:=toadd
else
if wheretoadd^.info>=toadd^.info then
begin
toadd^.next:=wheretoadd;
wheretoadd:=toadd;
end
else
```

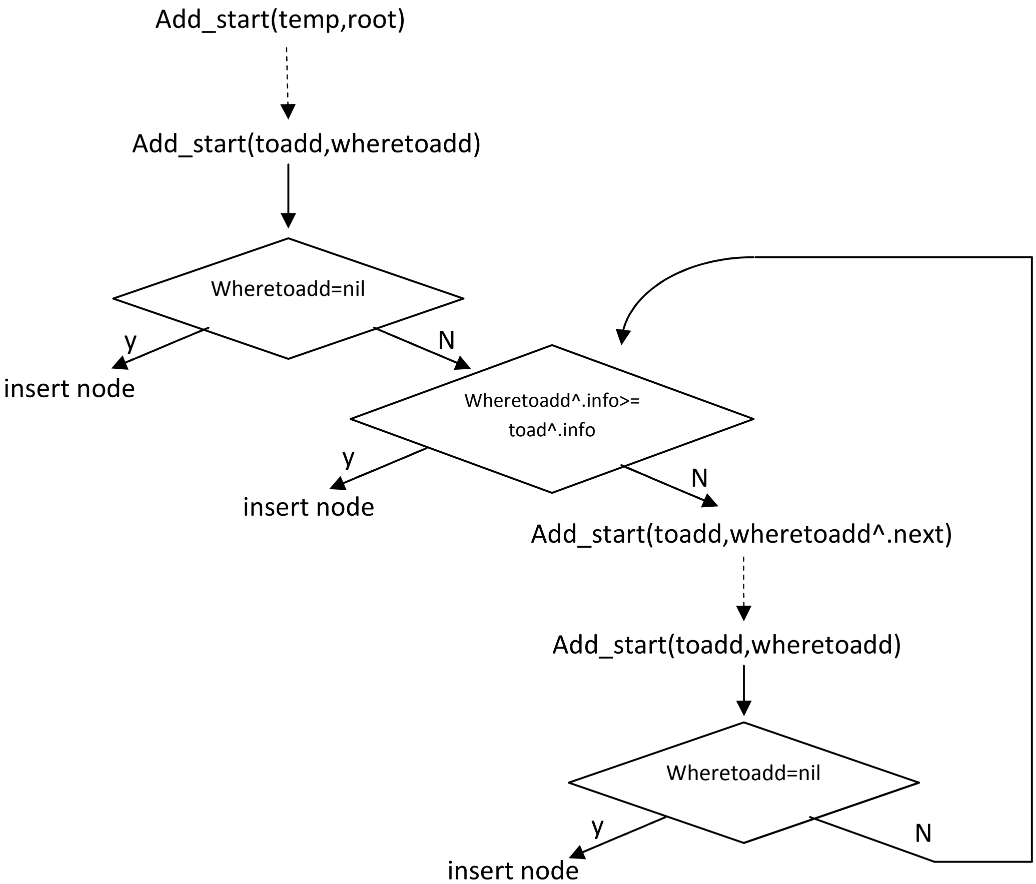
```
add_start(toadd,wheretoadd^.next);
end;
```

هذا الإجراء يشبه الإجراءين السابقين فهو يتحرك ضمن اللائحة مقارناً قيم الحقل info للعقد باحثاً عن المكان المناسب للعقدة الجديدة. فإذا كانت اللائحة فارغة empty أو وصل الإجراء إلى نهاية اللائحة المترابطة فإن العقدة الجديدة سوف تضاف في نهاية اللائحة المترابطة كما هو الحال في الإجراء add_back .

أما إذا كانت قيمة الحقل info لعقد اللائحة أكبر من القيمة الجديدة فأن العقدة الجديدة سوف تضاف إلى مقدمة اللائحة المترابطة كما في الإجراء add_front .

وبعد أن تضاف العقدة الجديدة فإن القائمة الجديدة سوف تعدل و سوف تمرر وفق تسلسل الاستدعاءات حتى تصل إلى البرنامج الرئيسي الذي سوف يستقبل هذه اللائحة المعدلة .

ويبين الشكل التالي المخطط المنطقي للعمليات التي تتم ضمن add_start:



مثال:ترتيب أسطر من ملف example: sorting lines from a file:

افترض أننا نريد قراءة محتويات ملف ما ومن ثم إعادة كتابة هذه المحتويات و لكن وفق ترتيبها الأبجدي حيث تنقل محتويات الملف إلى لائحة مترابطة مرتبة أبجدياً ثم تعاد إلى نفس الملف مرتبة أبجدياً مع مراعاة حذف العنصر المنقول مباشرة .
سوف يرينا البرنامج التالي كيفية القيام بهذا العمل باستخدام اللوائح المترابطة:

code

```
program test;
type
str=string[80];
nodeptr=^node;
node=record
info:str;
next:nodeptr;
end;
var root,temp:nodeptr;
source:text;
sname:string;
nrlines:integer;
function fopened(var thefile:text; fname:string):boolean;
begin
assign(thefile,fname);
(*$I-*)
reset(thefile);
(*$I+*)
if IOResult=0 then
fopened:=true
else
fopened:=false
end;
function fmade(var thefile:text; fname:string):boolean;
begin
assign(thefile,fname);
(*$I-*)
rewrite(thefile);
(*$I+*)
if IOresult=0 then
fmade:=true
else
fmade:=false
end;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
procedure showprogress(val,small,large:longint);
const markerch='.';
begin
if (val mod small=(small-1))then
```

```
write(markerch);
if(val mod large=(large-1))then
writeln;
end;
procedure initnode(var thenode:nodeptr);
begin
new(thenode);
thenode^.info:='';
thenode^.next:=nil
end;
procedure add_start(toadd:nodeptr; var wheretoadd:nodeptr);
begin
if(wheretoadd=nil)then
wheretoadd:=toadd
else
if(wheretoadd^.info>=toadd^.info)then
begin
toadd^.next:=wheretoadd;
wheretoadd:=toadd;
end
else
add_start(toadd,wheretoadd^.next);
end;
procedure writelist(var thefile:text;thenode:nodeptr);
var temp:nodeptr;
begin
while thenode<>nil do
begin
writeln(thefile,thenode^.info);
temp:=thenode;
thenode:=thenode^.next ;
dispose(temp);
end;
end;
begin {main}
root:=nil;
temp:=nil;
nrlines:=0;
writeln('memavail = ',memavail:10,' bytes');
getstring('source file? ',sname);
if fopened(source,sname)then
begin
while(not eof(source))and(maxavail>sizeof(node))do
begin
initnode(temp);
readln(source,temp^.info);
inc(nrlines);
showprogress(nrlines,5,250);
add_start(temp,root);
```

```
end;
close(source);

if fmade(source,sname) then
writelist(source,root)
else
writeln(sname,' not write');
close(source);
writeln;
writeln(nrlines:5,' lines processed');
writeln('memavail = ',memavail:10,' bytes');
end
else
writeln('file error');
readln;
end.
```

بفرض أن الملف المطلوب قراءته و ترتيب محتوياته أبجديا موجود في الدليل التالي c:\f.txt و كانت محتويات الملف كالتالي:

C:\f.txt
Syria Arabic
Arabic
Syria
Khaled yassin alsheikh
Welcome to Syria Arabic
Syria Syria

تصبح محتويات الملف النصي بعد التنفيذ كالتالي:

```
Arabic
Khaled yassin alsheikh
Syria
Syria Arabic
Syria Syria
Welcome to Syria Arabic
```

يقرأ البرنامج السابق أسطر من الملف النصي source و من ثم يضيف هذه الأسطر إلى لائحة مترابطة ترتب عقدها على أساس السلاسل الرمزية (أي محتويات الحقل info) ومن ثم يكتب البرنامج محتويات اللائحة المترابطة في نفس الملف مع حذف المساحة التخزينية التي خصصت للعنصر المنقول من الذاكرة (الكومة).

تم تحديد طول السلسلة الرمزية بما لا يتجاوز 80 رمزا على الأكثر و هذا التحديد سوف يقلل من استهلاك ذاكرة الكومة و بالتالي سوف يمكننا من تخصيص عقداً أكثر في ذاكرة الكومة.

من خلال البرنامج السابق نلاحظ ما يلي:

١. أن روتينات معالجة اللائحة المترابطة تستدعى في حلقة while في البرنامج الرئيسي حيث يخصص البرنامج مساحة من الذاكرة لكل سطر جديد باستخدام initnode.
٢. ثم تضاف مساحة الذاكرة تلك إلى اللائحة المترابطة بواسطة الإجراء add_start.
٣. سوف تكتب الأسطر المرتبة في نفس الملف source بعد فتحه للكتابة باستخدام الإجراء writelist و تحذف المساحات المخصصة للعقد من الذاكرة (الكومة).

مثال:استخدام عقد أكثر تعقيداً example: using more complex nodes:

إن التعقيدات التي يمكن إضافتها على عقد اللائحة المترابطة تعتمد و بشكل أساسي على أمرين اثنين:أولهما احتياجات البرنامج و ثانيهما مقدرة المبرمج نفسه على تخيل هذه العقد.

لقد استخدمنا تعريفاً للعقدة node في البرامج السابقة يستخدم لبناء اللوائح المترابطة الأحادية singly linked lists و هذا يعني أن كل عقدة في هذه اللائحة تشير إلى عقدة تالية عدا العقدة الأخيرة التي تشير إلى اللاشيء nil .

يمكننا إنشاء لوائح مترابطة مضاعفة doubly linked list هذا يعني كل عقدة في اللائحة المترابطة المضاعفة يمكن أن تشير إلى العقدة التالية و إلى العقدة السابقة في اللائحة.

وفيما يلي تعريف لعقد اللائحة المترابطة المضاعفة:

```
Type
dlptr=^dlist;
dlist=record
info:real;
next,preceding:dlptr;
end;
```

تختلف اللائحة المترابطة المضاعفة عن سابقتها الأحادية في أن عبورها traverse يمكن أن يتم بالاتجاهين لأن وجود مؤشرين في سجل واحد يعطينا مرونة كبيرة في طريقة التنقل.

فمثلاً يمكننا استخدام المؤشرين لبناء لائحتين مختلفتين من نفس العقد طبعا بالاعتماد على معلومات مختلفة ضمن هاتين اللائحتين. و في هذه الحالة سوف يستخدم كل مؤشر لغرض مختلف.

المطلوب كتابة برنامج بلغة باسكال يقوم بقراءة محتويات ملف نصي source ومن ثم يقوم بطباعة معلومات في ملف نصي آخر log حيث يتم كتابة المعلومات التالية الكلمة و ترددها(أي عدد مرات وردها في الملف) حسب الترتيب الأبجدي أو حسب ترتيب ترددها أو حسب ترتيبها الأبجدي و الترتيب الترددي. و المطلوب تحقيق المطلوب باستخدام مفهوم السلاسل المضاعفة.

Code

```
program test;
const emptystring='';
type
charset=set of char;
str=string[30];
dlnodeptr=^dlnode;
dlnode=record
wd:str;
freq:integer;
strnext,freqnext:dlnodeptr;
end;
var freqroot,root,temp:dlnodeptr;
currstr,currwd ,sname,lname:string;
source,log:text;
nrwds,nrunique:longint;
choice:integer;
procedure removewd(var wd,sentence:string);
const space=' ';
var where:integer;
begin
while pos(space,sentence)=1 do
delete(sentence,1,1);
where:=pos(space,sentence);
if where>0 then
begin
wd:=copy(sentence,1,where-1);
```

```
delete(sentence,1,where);
end
else
begin
wd:=sentence;
sentence:=emptystring;
end;
end;
function fopened(var thefile:text; fname:string):boolean;
begin
assign(thefile,fname);
(*$I-*)
reset(thefile);
(*$I+*)
if IOResult=0 then
fopened:=true
else
fopened:=false
end;
function fmade(var thefile:text; fname:string):boolean;
begin
assign(thefile,fname);
{$I-}
rewrite(thefile);
{$I+}
if IOResult=0 then
fmade:=true
else
fmade:=false
end;
procedure getstring(message:string; var value:string);
begin
write(message,' ');
readln(value);
end;
procedure showprogress(val,small,large:longint);
const markerch='.';
begin
if (val mod small=(small-1))then
write(markerch);
if (val mod large=(large-1))then
writeln;
end;
procedure makestrlower(var thestr:string);
const offset=32;
var index,howlong:integer;
begin
howlong:=length(thestr);
for index:=1 to howlong do
```

```
begin
if thestr[index] in ['A'..'Z'] then
inc(thestr[index],offset);
end;
end;
procedure cleanstr(var thestr:string; theset:charset);
const space=' ';
var index,howlong:integer;
begin
howlong:=length(thestr);
for index:=1 to howlong do
begin
if not(thestr[index] in theset)then
thestr[index]:=space;
end;
end;
procedure getboundedint(message:string;minval,maxval:integer;var value:integer);
procedure adjustvals(var small,large:integer);
var temp:integer;
begin
if small>large then
begin
temp:=small;
small:=large;
large:=temp;
end;
end;
function isbetween(val,small,large:integer):boolean;
begin
isbetween:=(small<=val)and(large>=val);
end;
begin
adjustvals(minval,maxval);
repeat
write(message,' ');
readln(value);
until isbetween(value,minval,maxval);
end;
procedure initdlnode(var thenode:dlnodeptr);
begin
new(thenode);
thenode^.wd:=emptystring;
thenode^.freq:=0;
thenode^.freqnext:=nil;
thenode^.strnext:=nil;
end;
procedure Iterfwritedlist(var thefile:text; thenode:dlnodeptr);
var count:integer;
begin
```

```
count:=0;
while thenode<>nil do
begin
writeln(thefile,thenode^.wd,' : ',thenode^.freq);
inc(count);
thenode:=thenode^.strnext;
end;
writeln(count:4,' entries in word list');
end;
procedure iiterfwritedlist(var thefile:text;thenode:dlnodeptr);
var count:integer;
begin
count:=0;
while thenode<>nil do
begin
writeln(thefile,thenode^.freq:4,' : ',thenode^.wd);
inc(count);
thenode:=thenode^.freqnext;
end;
writeln(count:4,' entries in frequency list');
end;
procedure add_start1(toadd:dlnodeptr; var wheretoadd:dlnodeptr);
begin
if wheretoadd=nil then
wheretoadd:=toadd
else
if wheretoadd^.wd>toadd^.wd then
begin
toadd^.strnext:=wheretoadd;
wheretoadd:=toadd;
end
else
if(wheretoadd^.wd=toadd^.wd)then
begin
inc(wheretoadd^.freq);
end
else
add_start1(toadd,wheretoadd^.strnext);
end;
procedure add_start2(toadd:dlnodeptr;var wheretoadd:dlnodeptr);
begin
if wheretoadd=nil then
wheretoadd:=toadd
else
if wheretoadd^.freq<=toadd^.freq then
begin
toadd^.freqnext:=wheretoadd;
wheretoadd:=toadd;
end
```

```
else
add_start2(toadd,wheretoadd^.freqnext);
end;
begin {main program}
writeln('memavail = ',memavail:10,' bytes');
writeln('maxavail = ',maxavail:10,' bytes');
root:=nil;
freqroot:=nil;
temp:=nil;
nrwds:=0;
nrunique:=0;
getstring('source file? ',sname);
getstring('log file? ',lname);
if fopened(source,sname)and fmade(log,lname) then
begin
while(not eof(source)and (maxavail>sizeof(dlnode)))do
begin
readln(source,currstr);
makestrlower(currstr);
cleanstr(currstr,['a'..'z','A'..'Z','-','''']);
repeat
removewd(currwd,currstr);
if (currwd<>emptystring)then
begin
inc(nrwds);
showprogress(nrwds,5,250);
initdlnode(temp);
temp^.wd:=currwd;
temp^.freq:=1;
add_start1(temp,root);
end;
until currstr=emptystring;
end;
writeln;
writeln(nrwds:10,' words');
temp:=root;
while(temp<>nil)do
begin
add_start2(temp,freqroot);
temp:=temp^.strnext;
end;
getboundedint('save by 1) word; 2) frequency; 3) both',1,3,choice);
case choice of
1:iterfwritedlist(log,root);
2:iiterfwritedlist(log,freqroot);
3:begin
iterfwritedlist(log,root);
writeln(log,'-----------------------------------');
iiterfwritedlist(log,freqroot);
```

```
end;
end;
writeln('memavail = ',memavail:10,' bytes');
writeln('maxavail = ',maxavail:10,' bytes');
close(source);
close(log);
end
else
writeln('file error.');
readln;
end.
```

شرح البرنامج:

يبني البرنامج السابق لائحة الكلمات أولاً بتخصيص مساحات من الذاكرة لكل عقدة فمن أجل كل كلمة يقرأها البرنامج من الملف source يستدعي الإجراء initdlnode و بعد بعض العمليات يضيف البرنامج هذه العقدة التي تحتوي الكلمة إلى لائحة الكلمات و ذلك باستخدام الإجراء add_start1 و قد أضفنا جزء else في عبارة if حيث يعالج هذا الجزء حالة كون الكلمة الجديدة مطابقة تماماً لكلمة موجودة في اللائحة حيث تفحص هذه الحالة باستخدام التعبير التالي:

Wheretoadd^.wd=toadd^.wd

فإذا كانت قيمة التعبير true فإن التعليمة لن تضاف و لكن سوف تزداد قيمة ترددها بمقدار 1 . وبما أن هذه العقدة لن تُضاف إلى اللائحة فإن مساحة التخزين المخصصة لها سوف تفقد عندما يستدعى البرنامج الإجراء initidlnode مرة أخرى و سوف نخسر مساحة تخزينية من الكومة.

يمكننا تعديل البرنامج السابق حتى يعيد مساحات التخزين غير المستخدمة و المستخدمة و هذا التعديل سوف يمكننا من معالجة عدد أكبر من الكلمات المختلفة.

التعديل الأول من أجل إعادة مساحات التخزين غير المستخدمة سيكون في الإجراء الذي يبنى لائحة الكلمات add_start1 و هذا التعديل بسيط جدا و هو أن نضع dispose (toadd) في المكان المناسب كما يلي:

Code

```
procedure add_start1(toadd:dlnodeptr; var wheretoadd:dlnodeptr);
begin
if wheretoadd=nil then
wheretoadd:=toadd
else
if wheretoadd^.wd>toadd^.wd then
begin
toadd^.strnext:=wheretoadd;
wheretoadd:=toadd;
end
else
if(wheretoadd^.wd=toadd^.wd)then
begin
inc(wheretoadd^.freq);
dispose(toadd);
end
```

```
else
add_start1(toadd,wheretoadd^.strnext);
end;
```

التعديل الثاني يعيد مساحات التخزين التي تم استهلاكها في الذاكرة (الكومة) بعد انتهاء تنفيذ البرنامج و التعديل يكون في الإجراء كما يلي:

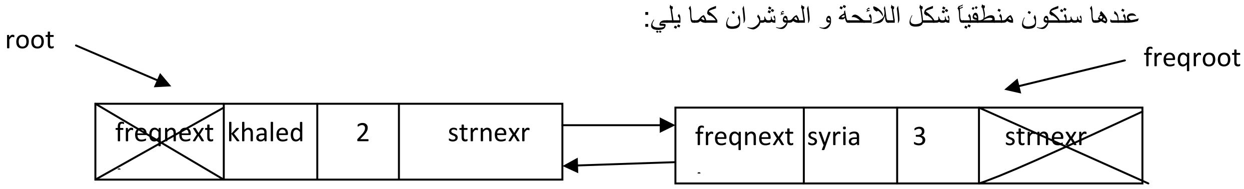
Code

```
procedure iiterfwritedlist(var thefile:text; var thenode:dlnodeptr);
var count:integer;
del:dlnodeptr;
begin
count:=0;
while thenode<>nil do
begin
writeln(thefile,thenode^.freq:4,' : ',thenode^.wd);
del:=thenode;
inc(count);
thenode:=thenode^.freqnext;
dispose(del);
end;
writeln(count:4,' entries in frequency list');
end;
```

تبنى لائحة الترددات و التي دليلها هو freqroot كما يلي: (بينما دليل لائحة الكلمات هو root)
يبني الإجراء add_start2 لائحة الترددات التي دليلها freqroot بالترتيب التنازلي لقيمة تردد الكلمة و نلاحظ أن المؤشر freqroot لا علاقة له بالمؤشر root لأن لائحة الكلمات مستقلة عن لائحة الترددات و حتى ولو كانت اللائحتان تستخدمان نفس العقد. وسأوضح ذلك من خلال الرسم:
بفرض أن محتويات الملف النصي source هي كالتالي:

C:\f.txt
Khaled Syria
Syria khaled
syria

عندها ستكون منطقياً شكل اللائحة و المؤشران كما يلي:



بفرض أن محتويات الملف النصي source كما يلي :

C:\f.txt
Syria Arabic in Damascus
khaled yassin alsheikh
damascus arabic arabic
yassin khaled khalED
welcome to syriA arabic
good bye;;

عندها تكون محتويات الملف النصي log كالتالي: بفرض اخترنا الرقم 1:

```
alsheikh : 1
arabic : 4
bye : 1
damascus : 2
good : 1
in : 1
khaled : 3
syria : 2
to : 1
welcome : 1
yassin : 2
```


تكون محتويات الملف النصي log كالتالي: بفرض اخترنا الرقم 2:

```
   4 : arabic
   3 : khaled
   2 : yassin
   2 : syria
   2 : damascus
   1 : welcome
   1 : to
   1 : in
   1 : good
   1 : bye
   1 : alsheikh
```


و تكون محتويات الملف النصي log كالتالي: بفرض اخترنا الرقم 3:

```
alsheikh : 1
arabic : 4
bye : 1
damascus : 2
good : 1
in : 1
khaled : 3
syria : 2
to : 1
welcome : 1
yassin : 2
-----------------------------------
   4 : arabic
   3 : khaled
   2 : yassin
   2 : syria
   2 : damascus
   1 : welcome
   1 : to
   1 : in
   1 : good
   1 : bye
   1 : alsheikh
```

اشرح التعريف التالي:

Var pt:^T;

أي أن المتحول PT هي مؤشر ^ إلى متحولات من النمط T و للوصول إلى المتحول الذي يشير إليه PT نستخدم الكتابة PT^
إذ يمكن الكتابة ;PT:^T <==> PT^ ∋ T.

لا تنسى أبداً أن المؤشرات هي عبارة عن عناوين في الذاكرة.

اكتب برنامج بلغة باسكال القياسية يقوم بطباعة عناصر لائحة خطية وحيدة الاتجاه بشكل معكوس في ملف نصي t.txt مع مراعاة حذف العنصر المنقول علما أن عنصر السلسلة يحوي حقل للمعطيات و هو القيمة v من نمط صحيح عبارة عن رقم الموظف.

سنتعرف الآن على بنى المعطيات الخطية باستخدام المؤشرات ومن أشهرها: المكدس stack والطابور(الرتل) Queue

(طبعاً ليس الطابور الخامس ☺ ☺ ☺).

المكدس Stack:

المكدس من أشهر بنى المعطيات الخطية المستخدمة بكثرة في العديد من التطبيقات البرمجية و الشبكية حيث أن عمليات الإضافة push و الحذف pop و البحث تتم من جهة واحدة هي قمة المكدس top.

ويعرف باسم last in First Out (LIFO).

سأقوم الآن بكتابة الإجرائيات الأساسية و بنية المعطيات عند التعامل مع المكدس stack بافتراض وجود حقل للمعطيات من النمط صحيح.

Code

```
program test;
type
pstack=^stack;
stack=record
val:integer;
next:pstack;
end;
```

إجرائية push لإضافة عنصر في قمة المكدس.

```
procedure push(var p:pstack; x:integer);
var temp:pstack;
begin
if (p=nil)then
begin
new(p);
p^.val:=x;
p^.next:=nil;
end
else
begin
new(temp);
temp^.val:=x;
temp^.next:=p;
p:=temp;
end;
end;
```

إجرائية pop لحذف عنصر (عقدة) من قمة المكدس.

```
procedure pop(var p:pstack);
var temp:pstack;
```

```
begin
if (p<>nil)then
begin
temp:=p;
p:=p^.next;
dispose(temp);
end;
end;
```

إجرائية top لإعادة عنصر من قمة المكدس.

```
procedure top(p:pstack; var res:integer);
begin
if p<>nil then
res:=p^.val;
end;
or:
function top(p:pstack):integer;
begin
if p<>nil then
top1:=p^.val;
end;
```

إجرائية del_all لتفريغ المكدس (حف المساحات التخزينية التي خصصت لعقد اللائحة من الذاكرة (الكومة)).

```
procedure del_all(var p:pstack);
begin
while p<>nil do
pop(p);
end;
```

تأهيل(تهيئة) المكدس.

```
procedure format(var p:pstack);
begin
p:=nil;
end;
```

السؤال :

لدينا ملف نصي f باللغة الإنكليزية و المطلوب كتابة برنامج يقوم بعكس أسطر الملف مستخدماً بنية المكدس مع مراعاة حذف العنصر الذي يتم طباعته مباشرة حيث يُراد طباعة الملف النصي f على شاشة الحاسب و ضمن ملف نصي f2.

Code

```
program test;
type
pstack=^stack;
stack=record
val:char;
next:pstack;
end;
var f2:text;
procedure format(var p:pstack);
begin
p:=nil;
end;
procedure pop(var p:pstack);
```

```
var e:pstack;
begin
if p<>nil then
begin
e:=p;
p:=p^.next;
dispose(e);
end;
end;
procedure top(var p:pstack; var f2:text);
begin
while p<>nil do
begin
write(p^.val);
write(f2,p^.val);
pop(p);
end;
writeln;
writeln(f2);
end;


procedure push(var p:pstack; c:char);
var temp:pstack; f:text;
begin
assign(f,'c:\f.txt');
reset(f);
while(not eof(f))do
begin
format(P);
while(not eoln(f))do
begin
read(f,c);
new(temp);
temp^.next:=nil;
temp^.val:=c;
if p=nil  then
p:=temp
else
begin
temp^.next:=p;
p:=temp
end
end;
top(p,f2);
readln(f);
end;
close(f);
end;
```

```
var head:pstack;
c:char;
begin
assign(f2,'c:\f2.txt');
rewrite(f2);
format(head);
push(head,c);
close(f2);
readln;
end.
```

C:\f.txt
Syria Arabic in Damascus khaled yassin alsheikh damascus   arabic arabic yassin khaled khalED welcome to syriA   arabic syria syria syria syria arab republic

تصبح محتويات الملف f2.txt كالتالي:

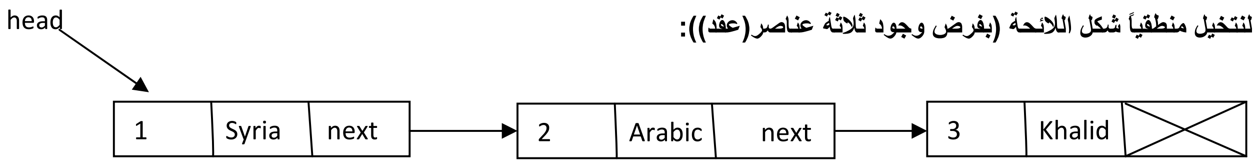
```
sucsamaD ni cibarA airyS
hkiehsla nissay  delahk
cibara  cibara   sucsamad
DElahk delahk nissay
cibara   Airys ot emoclew
airys airys airys
cilbuper bara airys
```

ويُطبع على الشاشة أيضاً:

```
sucsamaD ni cibarA airyS
hkiehsla nissay  delahk
cibara  cibara   sucsamad
DElahk delahk nissay
cibara   Airys ot emoclew
airys airys airys
cilbuper bara airys
```

السؤال:
اكتب برنامج بلغة باسكال القياسية يقوم بطباعة عناصر لائحة خطية وحيدة الاتجاه بشكل معكوس في ملف نصي t.txt مع مراعاة حذف العنصر المنقول علما أن عنصر السلسلة يحوي حقل للمعطيات و هو القيمة ID من نمط صحيح عبارة عن رقم الموظف و حقل المعطيات name من نمط سلسلة رمزية string.
الحل:

لنتخيل منطقياً شكل اللائحة (بفرض وجود ثلاثة عناصر(عقد)):



يُراد نقل عناصر اللائحة إلى ملف نصي بشكل معكوس . (الحل باستخدام المكدس):

Code

```
program test;
type
pstack=^stack;
stack=record
id:integer;
name:string[30];
next:pstack;
end;
var f2:text;
procedure format(var p:pstack);
begin
p:=nil;
end;
procedure pop(var p:pstack);
var e:pstack;
begin
if p<>nil then
begin
e:=p;
p:=p^.next;
dispose(e);
end;
end;
procedure top(var p:pstack; var f2:text);
begin
while p<>nil do
begin
writeln(f2,p^.id:3,' ':5,p^.name:10);
writeln(p^.id:3,' ':5,p^.name:10);
pop(p);
end;
end;
procedure push(var p,wheretoadd:pstack);
var temp,e:pstack; f:text;
```

```
begin
temp:=wheretoadd;
if p=nil then
begin
new(p);
p^:=temp^;
p^.next:=nil;
wheretoadd:=wheretoadd^.next;
dispose(temp);
end
else
begin
new(e);
e^:=temp^;
wheretoadd:=wheretoadd^.next;
dispose(temp);
e^.next:=p;
p:=e;
end;
end;
procedure add_start1(toadd:pstack;var wheretoadd:pstack);
begin
if (wheretoadd=nil)then
wheretoadd:=toadd
else
if(wheretoadd^.id>toadd^.id)then
begin
toadd^.next:=wheretoadd;
wheretoadd:=toadd;
end
else
add_start1(toadd,wheretoadd^.next);
end;
var head,root,temp:pstack;
c:char;
begin
assign(f2,'c:\f.khaled');
rewrite(f2);
format(head);
format(root);
while(not eof)do  {'press Ctrl+Z for end input data'}
begin
new(temp);
temp^.next:=nil;
readln(temp^.id);
readln(temp^.name);
add_start1(temp,head);
end;
while(head<>nil)do
```

```
push(root,head);
top(root,f2);
close(f2);
readln;
end.
```

بفرض أننا أدخلنا عناصر العقد السابقة بالترتيب يكون تنفيذ البرنامج هو:

```
1
syria
2
arabic
3
khalid
^Z
  3          khalid
  2          arabic
  1           syria
```

و بالتالي تصبح محتويات الملف النصي c:\f.khaled كما يلي:

```
3            khalid
2            arabic
1             syria
```

السؤال:

مستخدماً بنية المكدس قم بكتابة الإجرائيات اللازمة لتحويل عدد عشري صحيح موجب إلى ما يكافئه بالثنائي.
مثال: العدد $(250)_{10}$ يكافئه العدد الثنائي $(11111010)_2$.
طريقة التحويل :

باقي القسمة ↓

250	2	→	0	→ 11111010
125	2	→	1	↑
62	2	→	0	↑
31	2	→	1	↑
15	2	→	1	↑
7	2	→	1	↑
3	2	→	1	↑
1	2	→	1	↑
0				

Code

```
program test;
type
pstack=^stack;
stack=record
id:0..1;
next:pstack;
end;
procedure format(var p:pstack);
begin
p:=nil;
end;
procedure pop(var p:pstack);
var e:pstack;
begin
if p<>nil then
begin
e:=p;
p:=p^.next;
dispose(e);
end;
end;
procedure top(var p:pstack);
begin
while p<>nil do
begin
write(p^.id);
pop(p);
end;
end;
procedure push(var p:pstack;x:integer);
```

```
var temp:pstack;
begin
if (x<>0)then
begin
if p=nil then
begin
new(p);
p^.id:=x mod 2;
p^.next:=nil;
end
else
begin
new(temp);
temp^.id:=x mod 2;
temp^.next:=p;
p:=temp;
end;
push(p,x div 2);
end
end;
var head,root,temp:pstack;
x:integer;
begin
format(root);
readln(x);
push(root,x);
top(root);
readln;
end.
```

عدل البرنامج السابق بحيث يتم تحويل العدد العشري الصحيح الموجب إلى ما يكافئه بالثماني.

الطابور (الرتل) Queue:

الرتل أو الطابور من أشهر بنى المعطيات الخطية المستخدمة بكثرة في التطبيقات البرمجية و الشبكية. حيث أن عمليات الإضافة تتم في ذيل(نهاية) الرتل rear و عملية الحذف (التخديم) تتم في بداية لرتل front. وبعرف باسم First In First Out(FIFO) .

سأقوم الآن بكتابة الإجرائيات الأساسية و بنية المعطيات عند التعامل مع الطابور Queue بافتراض وجود حقل للمعطيات من النمط صحيح. بطريقتين:

الطريقة الأولى:

Code

```
program test;
type
pQueue=^Queue;
Queue=record
ID:integer;
next:pQueue;
end;
Queue_rec=record
front,rear:pQueue;
end;
```

إجرائية EnQueue لإضافة عنصر(عقدة) إلى ذيل(نهاية) الطابور.

```
procedure EnQueue(var Q:Queue_rec; x:integer);
var temp:pQueue;
begin
new(temp);
temp^.id:=x;
temp^.next:=nil;
if Q.front=nil then
begin
Q.rear:=temp;
Q.rear:=temp;
end
else
begin
Q.rear^.next:=temp;
Q.rear:=temp;
end
end;
```

إجرائية DeQueue لحذف عنصر من بداية الطابور(الرتل).

```
procedure DeQueue(var Q:Queue_Rec);
var temp:pQueue;
begin
if Q.front<>nil then
begin
temp:=Q.front;
Q.front:=Q.front^.next;
dispose(temp);
end;
end;
```

إجرائية del_Queue لتفريغ الرتل (الطابور).

```
procedure del_Queue(var Q:Queue_Rec);
begin
while(Q.front<>nil)do
DeQueue(Q);
end;
```

إجرائية print_Queue لطباعة عناصر(عقد) الرتل.

```
procedure print_Queue(Q:Queue_Rec);
begin
while(Q.front<>nil)do
begin
write(Q.front^.id,' ');
Q.front:=Q.front^.next;
end
end;
```

تابع replay لإعادة عنصر من بداية الرتل.

```
function replay(Q:Queue_rec):integer;
begin
if Q.front<>nil then
replay:=Q.front^.ID;
end;
begin
readln
end.
```

الطريقة الثانية:

Code

```
program test;
type
pQueue=^Queue;
Queue=record
id:integer;
next:pQueue;
end;
procedure EnQueue(var first,last:pQueue; x:integer);
var temp:pQueue;
begin
new(temp);
temp^.id:=x;
temp^.next:=nil;
if(first=nil)then
begin
first:=temp;
last:=first;
end
else
begin
last^.next:=temp;
```

```
last:=temp
end
end;
procedure DeQueue(var first:pQueue);
var temp:pQueue;
begin
if (first<>nil)then
begin
temp:= first;
first:=first^.next;
dispose(temp);
end
end;
procedure del_Queue(var first:pQueue);
begin
while(first<>nil)do
DeQueue(First);
end;
procedure format(var first:pQueue);
begin
first:=nil;
end;
begin
readln
end.
```

السؤال:

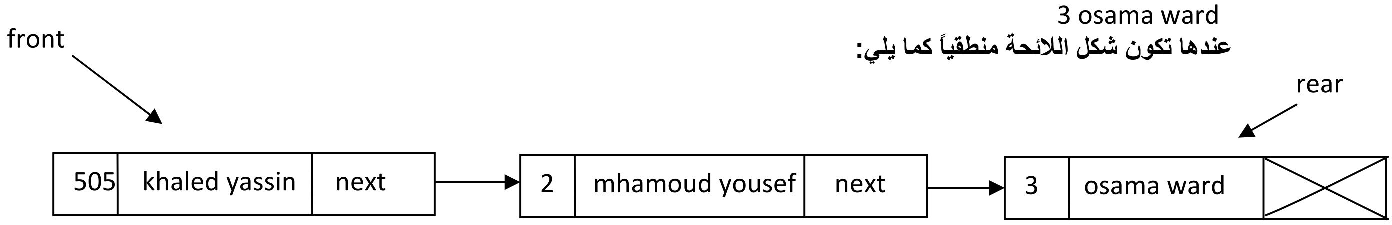
تحقيقاً لمبدأ النافذة الواحدة لتخديم مواطني الجمهورية العربية السورية بمراكز الإدارة المحلية حيث يقوم المواطنين بتسجيل أرقامهم و اسماهم ضمن لوحة إلكترونية مؤتمتة حيث يتم تخديم المواطنين بتسلسل تسجيل الأسماء ضمن اللوحة الإلكترونية المؤتمتة و المطلوب مستخدماً بنية الرتل (الطابور) قم بكتابة الإجرائيات اللازمة لطباعة أرقام المواطنين مع أسماهم ضمن قاعدة بيانات و لتكن هنا ملف نصي f و على شاشة الحاسب.

الحل:

مثل:بفرض أن الإدخال ضمن اللوحة تم وفق التسلسل التالي:

505 khaled yassin
2 mhamoud yousef
3 osama ward

عندها تكون شكل اللائحة منطقياً كما يلي:



Code

```
program test;
type
pQueue=^Queue;
Queue=record
id:integer;
name:string[30];
next:pQueue;
end;
Rec_Queue=record
front,rear:pQueue;
end;
var Q:Rec_Queue;  temp:pQueue;
f:text;
procedure EnQueue(toadd:pQueue; var Q:Rec_Queue);
begin
if Q.front=nil then
begin
Q.front:=toadd;
Q.rear:= toadd
end
else
begin
Q.rear ^.next:=toadd;
Q.rear:=toadd;
end;
end;
procedure DeQueue(var Q:Rec_Queue);
var  temp:pQueue;
begin

if (Q.front<>nil)then
begin
temp:=Q.front;
Q.front:=Q.front^.next;
writeln(f,temp^.id:3,' ':5,temp^.name:10);
writeln(temp^.id:3,' ':5,temp^.name:10);
dispose(temp);
end;
end;
procedure del_Queue(var Q:Rec_Queue);
begin
while(Q.front<>nil)do
DeQueue(Q);

end;
procedure format(var Q:Rec_Queue);
begin
Q.front:=nil;
```

```
Q.rear:=nil;
end;
begin
writeln(memavail);
assign(f,'c:\f.txt');
rewrite(f);
format(Q);
while(not eof)do
begin
new(temp);
readln(temp^.id);
readln(temp^.name);
temp^.next:=nil;
EnQueue(temp,Q);
end;
del_Queue(Q);
close(f);
writeln(memavail);
readln;
end.
```

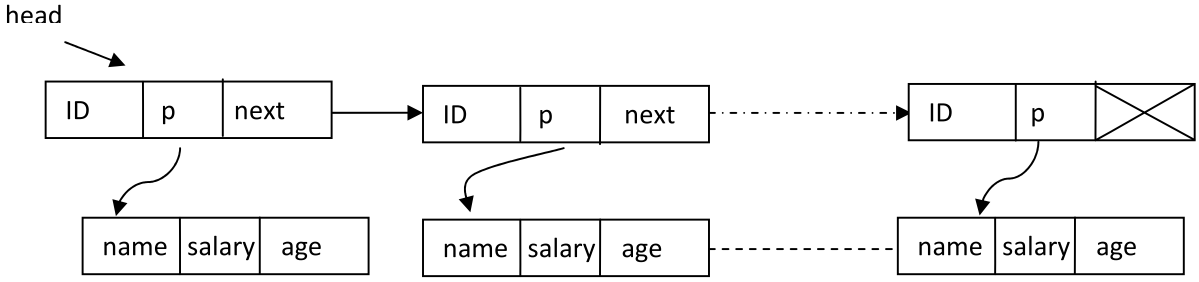
بفرض أننا أدخلنا عناصر العقد السابقة بالترتيب يكون تنفيذ البرنامج هو:

```
539424
505
khaled yassin
2
mahmoud yousef
3
osama ward
^Z
505      khaled yassin
  2      mahmoud yousef
  3      osama ward
539424
```


و بالتالي تصبح محتويات الملف C:\f.txt كما يلي:

```
505      khaled yassin
  2      mahmoud yousef
  3      osama ward
```

السؤال:
لدينا بيانات لموظفين مجلس الإدارة المحلية في مدينة معضمية الشام بدمشق و نريد تخزين هذه البيانات في بنية معطيات على الشكل المبين أدناه باستخدام سلسلة مترابطة من المؤشرات (لائحة) حيث ID هو رقم الموظف (لا يتكرر) وp مؤشر على تسجيلة تحوي name (اسم الموظف) و salary (راتب الموظف) و age (عمر الموظف).
علما أن عناصر السلسلة مرتبة تصاعديا حسب رقم الموظف.



و المطلوب:
١- عرف بنية معطيات تكون مناسبة لتمثيل شكل اللائحة أعلاه.
٢- إجرائية add_list لبناء لائحة مترابطة وفق الشكل السابق.
٣- إجرائية del_list لحذف جميع المساحات التخزينية المخصصة للعقد في الذاكرة (الكومة).
٤- إجرائية عودية print لطباعة عناصر كل عقدة على سطر على شاشة الحاسب.

Code
```
program test;
type
ptr=^info;
data=^page;
info=record
id:integer;
p:data;
next:ptr;
end;
page=record
name:string[30];
salary:real;
age:byte;
end;
procedure add_list(toadd:ptr; var wheretoadd:ptr);
begin
if wheretoadd=nil then
wheretoadd:=toadd
else
if(wheretoadd^.id>toadd^.id)then
begin
toadd^.next:=wheretoadd;
wheretoadd:=toadd;
end
else
add_list(toadd,wheretoadd^.next);
```

```
end;
procedure print(head:ptr);
begin
if head<>nil then
begin
writeln(head^.id:3,' ':5,head^.p^.name:10,' ':4,head^.p^.salary:8:2,' ':4,head^.p^.age:4);
print(head^.next);
end;
end;
procedure del_list(var head:ptr);
var temp:ptr;
begin
while(head<>nil)do
begin
temp:=head;
head:=head^.next;
dispose(temp^.p);
dispose(temp);
end;
end;
var head,temp:ptr;
begin
writeln('memory heap =',memavail);
head:=nil;
while(not eof)do  {press Ctrl+Z for end input Data}
begin
new(temp);
new(temp^.p);
temp^.next:=nil;
readln(temp^.id);
readln(temp^.p^.name);
readln(temp^.p^.salary);
readln(temp^.p^.age);
add_list(temp,head);
end;
writeln;
writeln('id':3,' ':5,'name':10,' ':4,'salary':8,' ':4,'age':4);
writeln;
print(head);
del_list(head);
writeln('memory heap = ',memavail);
readln;
end.
```

السؤال:

لدينا بيانات طلاب كلية الهندسة المعلوماتية بجامعة دمشق (السنة الأولى) مخزنة ضمن ملف نصي موجود فيزيائياً على القرص بالدليل c. info.kha و منطقياً ضمن البرنامج بالاسم f بحيث كل سطر من أسطر الملف تمثل بيانات طالب على الشكل التالي:

رقم الطالب   اسم الطالب الأول   اسم الطالب الثاني   مواليد الطالب

حيث يفصل بين الرقم و الاسم فراغ واحد على الأقل وكذلك الأمر بالنسبة لباقي البيانات في كل سطر.
و المطلوب نقل بيانات الطلاب إلى لائحة(سلسلة) خطية مناسبة. و من ثم نقل بيانات الطلاب مرتبة تصاعدياً حسب رقم الطالب إلى ملف ثنائي fb.kha مع مراعاة حذف العنصر المنقول مباشرة من الذاكرة(الكومة).
واكتب إجرائية عودية لطباعة عناصر اللائحة على شاشة الحاسب.

Code

```
program test;
type
ptr=^info;
info=record
ID:integer;
first_name:string[25];
last_name:string[25];
birthday:integer;
next:ptr;
end;
file_binary=file of info;
procedure mve_file_list(var head:ptr; var f:text);
var
temp,p1,p2:ptr;c:char; w:string;ok:boolean;
t:integer;
begin
assign(f,'c:\info.kha');
reset(f);
while(not eof(f))do
begin
t:=0;
w:='';
ok:=false;
new(temp);
temp^.next:=nil;
while(not eoln(f))do
begin
read(f,c);
if c<>' ' then
if(not(c in ['A'..'Z','a'..'z'])) then
t:=t*10+ord(c)-48
else
w:=w+c;
if(w<>'')and(c=' ')then
begin
if not ok then
```

```
begin
temp^.first_name:=w;
w:='';
end
else
begin
temp^.last_name:=w;
w:='';
end;
ok:=true;
end
else
if (t<>0) and (c=' ') or eoln(f)then
begin
if not ok then
begin
temp^.id:=t;
t:=0;
end
else
if eoln(f) then
begin
temp^.birthday:=t;
t:=0;
end
end
end;
if head=nil then
head:=temp
else
if head^.id>temp^.id then
begin
temp^.next:=head;
head:=temp;
end
else
begin
p1:=head;
while(p1<>nil)and(p1^.id<temp^.id)do
begin
p2:=p1;
p1:=p1^.next;
end;
p2^.next:=temp;
temp^.next:=p1;
end;
readln(f);
end;
close(f);
```

```
end;
procedure print( head:ptr);
begin
if (head<>nil)then
begin
writeln(head^.id:3,' ':5,head^.first_name:10,' ':4,head^.last_name:10,'
':4,head^.birthday:5);
print(head^.next);
end;
end;
procedure move_fb(var head:ptr; var f:file_binary);
var temp:ptr;
begin
assign(f,'c:\fb.kha');
rewrite(f);
while(head<>nil)do
begin
temp:=head;
head:=head^.next;
write(f,temp^);
dispose(temp);
end;
close(f);
end;
var f:text; head:ptr;
fb:file_binary;
begin
head:=nil;
writeln('memory heap = ',memavail);
writeln;
mve_file_list(head,f);
writeln('id':3,' ':5,'first_name':10,' ':4,'last_name':10,' ':4,'birthday':5);
writeln;
print(head);
move_fb(head,fb);
writeln('memory heap = ',memavail);
readln;
end.
```

بفرض أن محتويات الملف info.kha كالتالي:

```
3 khaled yassin 1900
1 ahmed jamoos 1800
5 osama ward 1700
4 haytham kabaz 1600
8 ali omar 1800
7 swosan jamal 1600
6 Iman alsheikh 1992
2 Ihab alqassim 1902
```

عند التنفيذ يكون تنفيذ البرنامج على شاشة الحاسب كالتالي:

```
memory heap = 537616
 id      first_name        last_name      birthday

  1           ahmed          jamoos        1800
  2            Ihab        alqassim        1902
  3          khaled          yassin        1900
  4         haytham           kabaz        1600
  5           osama            ward        1700
  6            Iman        alsheikh        1992
  7          swosan           jamal        1600
  8             ali            omar        1800
memory heap = 537616
```

المهندس خالد ياسين الشيخ

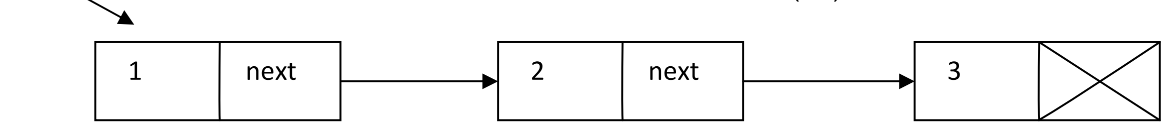
و كذلك سيكون الملف محدد النوع fb.kha مكون من 8 عناصر و كل عنصر عبارة عن سجل يحوي معلومات الطالب.
حجم السجل الواحد= 2+ 26 + 26 + 2 + 4 = 60 بايت و بالتالي حجم الملف الكلي= 8 × 60 = 480 bytes .
علماً أن المساحة التخزينية المخصصة من قبل مدير الكومة لبناء عقد اللائحة =(8+24+24+8)بايت × 8= 512 بايت

السؤال:

أكتب إجرائية reverse لعكس عناصر سلسلة (لائحة) خطية وحيدة الاتجاه **بدون** استخدام **عناصر** لائحة مساعدة.

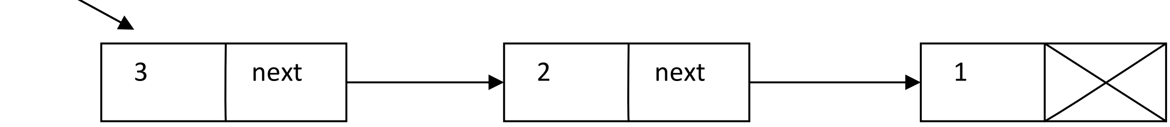
مثال: من أجل لائحة مؤلفة من ثلاث عناصر (عقد)

head



تصبح بعد التنفيذ على الشكل التالي:

head



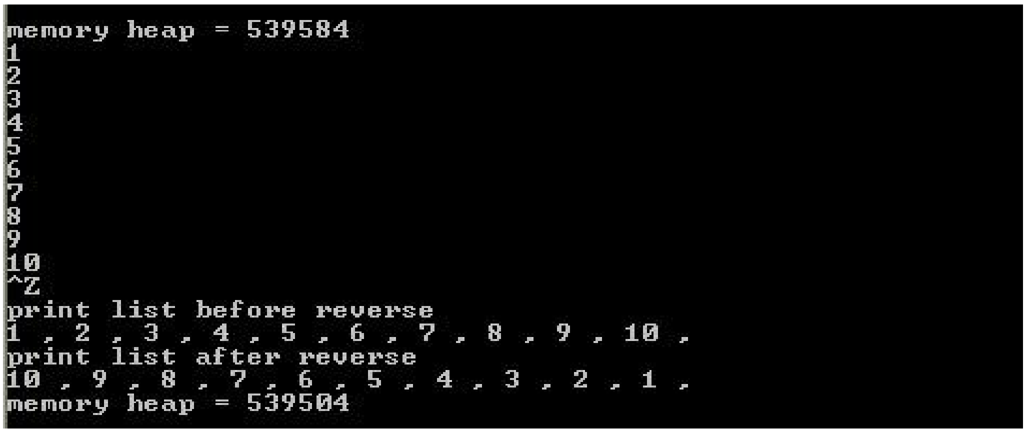
هناك الكثير من الطرق للحل ومنها:

Code

```
program test;
type
plist=^list;
list=record
id:integer;
next:plist;
end;
procedure add_list(toadd:plist; var wheretoadd:plist);
begin
if (wheretoadd=nil)then
wheretoadd:=toadd
else
if(wheretoadd^.id>toadd^.id)then
begin
toadd^.next:=wheretoadd;
wheretoadd:=toadd;
end
else
add_list(toadd,wheretoadd^.next);
end;
function length(head:plist):integer;
begin
if head=nil then
length:=0
else
length:=1+length(head^.next);
end;
procedure reverse(var head:plist);
var temp,p1,p2,head1:plist;
i:integer;
begin
for i:=1 to length(head) do
begin
```

```
p1:=head;
p2:=p1;
while(p1^.next<>nil)do
begin
p2:=p1;
p1:=p1^.next;
end;
p2^.next:=nil;
if i =1 then
head1:=p1
else
begin
temp:=head1;
while(temp^.next<>nil)do
temp:=temp^.next;
temp^.next:=p1;
end;
end;
head:=head1;
end;
procedure print(head:plist);
begin
if head<>nil then
begin
write(head^.id,' , ');
print(head^.next);
end;
end;
var head,temp:plist;
begin
head:=nil;
writeln('memory heap = ',memavail);
while(not eof)do  {press Ctrl+Z for end input data}
begin
new(temp);
readln(temp^.id);
temp^.next:=nil;
add_list(temp,head);
end;
writeln('print list before reverse');
print(head);
writeln;
reverse(head);
writeln('print list after reverse');
print(head);
writeln;
writeln('memory heap = ',memavail);
readln;
end.
```

إذا تم تعبئة السلسلة بالأرقام التالية على الترتيب : 1 و 2 و 3 و 4 و 5 و 6 و 7 و 8 و 9 و 10.
تكون نتائج تنفيذ البرنامج على شاشة الحاسب كالتالي:



استهلك البرنامج السابق مساحة تخزينية قدرها 10 × 8 بايت = 80 بايت من الذاكرة لأن غاية المؤشر في الكومة يُحجز له بشكل افتراضي 8 بايت.

اللهم صلي وسلم و بارك على سيدنا محمد و على آله و صحبه أجمعين

**محمد سيد الأعراب و العجم		محمد خير من يمشي على قدم**

**محمد باسط المعروف جامعه		محمد صاحب الإحسان و الكرم**

**محمد ذكره روح لأنفسنا		محمد شكره فرض على الأمم**

يقول الإمام الشافعي رحمه الله تعالى:

أحفظ لسانك أيها الإنسان		لا يلدغنك أنه ثعبانُ
كم في المقابر من قتيل لسانه		كانت تهاب لقاءه الشجعانُ
جراحات السنان لها التآم		ولا يلتام ما جرح اللسانُ

يقول محمد بن إدريس:

كلما أدبني الدهر		أراني نقص عقلي
و إذا ما ازددت علماً		زادني علما بجهلي

**أما خالد الشيخ فيقول:(ذهب و لم يعد)		٢٠١١/٦/١٤ دمشق–معضمية الشام**

لقاءنا في الجزء الثاني من مسلسل المؤشرات pointers
إخراج المهندس خالد ياسين الشيخ



ملاحظة على الهامش: أصعب شيء في الحياة هو أن لا تجد عمل(☠)